Sistemas y subsistemas de sustituciones explícitas

Arbiser, Ariel

2005

Tesis Doctoral

Explicit Substitution Systems and Subsystems

By Ariel Arbiser
Supervisor: Dr. Alejandro Ríos

Facultad de Ciencias Exactas y Naturales

Universidad de Buenos Aires

A thesis submitted for the degree of

Doctor of Philosophy in Computer Science

Jury:

Dr. Mauricio Ayala-Rincón, Dept. of Mathematics, Universidade de Brasília

Dra. Delia Kesner, PPS, Université Paris VII

Dra. Nora Szasz, Universidad ORT Uruguay

Submitted in partial fulfillment for the degree of Doctor of Philosophy in Computer Science. Second preliminary version.

Dedication

to my Family

Discovery is not the result of logical thought, even though the ultimate result is intimately bound to the rules of logic. – A. Einstein

The great advantage of the mathematical sciences above the moral consists in this, that the ideas of the former, being sensible, are always clear and determinate, the smallest distinction between them is immediately perceptible, and the same terms are still expressive of the same ideas, without ambiguity or variation. An oval is never mistaken for a circle, nor an hyperbola for an ellipsis. The isosceles and scalenum are distinguished by boundaries more exact than vice and virtue, right and wrong. If any term be defined in geometry, the mind readily, of itself, substitutes, on all occasions, the definition for the term defined: Or even when no definition is employed, the object itself may be presented to the senses, and by that means be steadily and clearly apprehended. But the finer sentiments of the mind, the operations of the understanding, the various agitations of the passions, though really in themselves distinct, easily escape us, when surveyed by reflection; nor is it in our power to recall the original object, as often as we have occasion to contemplate it. Ambiguity, by this means, is gradually introduced into our reasonings: Similar objects are readily taken to be the same: And the conclusion becomes at last very wide of the premises. – D. Hume, *An Enquiry Concerning Human Understanding*

Acknowledgement/Agradecimiento

Espero que este trabajo sirva para que otras personas puedan introducirse en el área de Reescritura, o conocer un poco más acerca de ella, y de modo que surjan problemas nuevos a partir de lo planteado aquí.

Un agradecimiento profundo y especial a mi familia, por diversas razones.

Quiero agradecer a Alejandro Ríos, mi director de tesis, quien fue quien me introdujo en el terreno de las sustituciones explícitas. Más allá de esto, Alejandro siempre encuentra tiempo para conversar sobre todo tema o idea que cae en nuestras manos.

Agradezco a Fairouz Kamareddine, coautora nuestra, con quien hemos conversado a distancia sobre interesantes cuestiones.

Agradezco a cada uno de los miembros del Jurado por la paciencia que implica esta lectura, y por sus sugerencias.

Mi agradecimiento también a Leonard, que fue algo así como un guía de facto en cierto momento, y quien insistió lo necesario para que me apurara un poco más.

Por último, por qué no, un agradecimiento al lector genérico que está por leer este texto, y corre el peligro de aburrirse prematuramente con los agradecimientos (incluyendo este mismo).

Finalizo deseando que esta tesis no sea una forma normal ... sino continuar (re)escribiendo dentro de este vasto campo, lleno de transformación, y de derivaciones infinitas ...

Abstract

This thesis is about some problems on rewriting theory and explicit substitution calculi. The main topic is the study of sub-calculi for several rewriting systems.

After a biased introduction to rewriting, λ-calculus and explicit substitution, we make a comparative study of the main rewriting formalisms, identifying a hierarchy between them.

As a first approach to new calculi, we address Revesz' λ-calculus with names, involving four rewriting rules, with the particularity that it does not have any substitution at all. We show the relative soundness and the confluence. We also propose and study two versions using de Bruijn indices, proving that all these properties are preserved.

We then move to perpetuality in the $\lambda\upsilon$ explicit substitution calculus, and study perpetual rewrite strategies, i.e. those strategies that preserve the possibility of infinite derivations. We give as an application a set of deterministic inference rules which characterize inductively the subsystem of strongly normalizing terms, and we present an effective perpetual reduction strategy for $\lambda\upsilon$.

Then we study different extensions of $\lambda\upsilon$-calculus, with the addition of composition-like rules. Weak confluence on open terms is proved. As an application, a derived simplification of $\lambda\upsilon$ with a unique de Bruijn index can be given, which is a sub-calculus of the former and has the same properties.

We show the weak normalization of the simply-typed λs_e-calculus with open terms where abstractions are decorated with types, and meta-variables, de Bruijn indices and updating operators are decorated with environments. The proof is based on the $\lambda\omega_e$-calculus, a calculus which over semi-open terms (i.e. those which allow term variables but not substitution variables) is isomorphic to λs_e over open terms. This proof is strongly influenced by a previous proof of weak normalization for the simply-typed $\lambda\sigma$-calculus but with subtle differences which show that the two styles require different attention. Furthermore, we give a new calculus, $\lambda\omega'_e$, sub-calculus of $\lambda\omega_e$, which handles a unique de Bruijn index, which is then closer to $\lambda\sigma$ than $\lambda\omega_e$. For $\lambda\omega'_e$ we also prove the weak normalization for typed semi-open terms.

We present an extension of the $\lambda(\eta)$-calculus with a primitive case construct that propagates through abstractions like a head linear substitution before doing constructor substitution, and show that this way of decomposing pattern matching allows to recover the expressiveness of ML-style pattern matching. Then we prove that this system enjoys confluence using a semi-automatic "divide and conquer" technique by which we determine all the pairs of commuting subsystems of the formalism (considering all the possible combinations of the nine reduction rules). Finally, we prove a (weak) separation theorem for the whole formalism, using a separation technique inspired by the Böhm-out technique.

And, as another facet of sub-calculi exploration, we investigate the terms which satisfy the property of expansion to pure terms, for $\lambda\upsilon$ and λs. We prove that these sets of terms are

proper and non-recursive. As an application, we prove the impossibility of adequate mappings between certain pairs of calculi.

Resumen

Esta tesis trata acerca de algunos problemas en la teoría de reescritura y cálculos con sustituciones explícitas. El tema principal es el estudio de sub cálculos de algunos sistemas de reescritura.

Luego de una introducción sesgada a la reescritura, el cálculo λ y las sustituciones explícitas, hacemos un estudio comparativo de los principales formalismos de la reescritura, identificando una jerarquía entre éstos.

Como primer aproximación a nuevos cálculos, estudiamos el cálculo λ con nombres de Revesz, que utiliza cuatro reglas de reescritura, con la particularidad de que no cuenta con sustitución alguna. Se prueba la correctitud relativa y la confluencia. También se proponen y se estudian dos versiones que usan índices de de Bruijn, y se prueba que estas dos propiedades se preservan.

Luego pasamos a la perpetualidad en el cálculo de sustituciones explícitas $\lambda\upsilon$ y estudiamos estrategias de reducción perpetuas, i.e. aquellas que preservan la posibilidad de derivaciones infinitas. Se da como una aplicación un conjunto de reglas de inferencia determinísticas que caracterizan inductivamente el sub sistema de los términos fuertemente normalizantes, y presentamos una estrategia de reducción perpetua efectiva para $\lambda\upsilon$.

A continuación se estudian distintas extensiones del cálculo $\lambda\upsilon$, con reglas al estilo de las de composición. Se prueba la confluencia débil sobre términos abiertos. Como aplicación de lo anterior, se puede dar un cálculo simplificado, derivado de $\lambda\upsilon$, que usa un solo índice de de Bruijn, el cual es un sub cálculo del anterior y con las mismas propiedades.

Se demuestra luego la normalización débil del cálculo λs_e simplemente tipado sobre términos abiertos, en donde las abstracciones se decoran con tipos, y las meta variables, índices de de Bruijn y operadores de actualización se decoran con contextos. La prueba se basa en el cálculo $\lambda\omega_e$, que sobre términos semi-abiertos (i.e. aquellos que admiten variables de término pero no de sustitución) es isomorfo a λs_e sobre términos abiertos. Esta prueba está fuertemente influenciada por otra previa de normalización débil para el cálculo $\lambda\sigma$ simplemente tipado pero con diferencias substanciales que indican que los dos estilos requieren distinto tratamiento. Además, introducimos el cálculo $\lambda\omega'_e$, sub cálculo de $\lambda\omega_e$, el cual usa sólo un índice de de Bruijn, con lo que es más cercano a $\lambda\sigma$ que $\lambda\omega_e$. Para $\lambda\omega'_e$ tipado probamos también la normalización débil sobre términos tipados semi-abiertos.

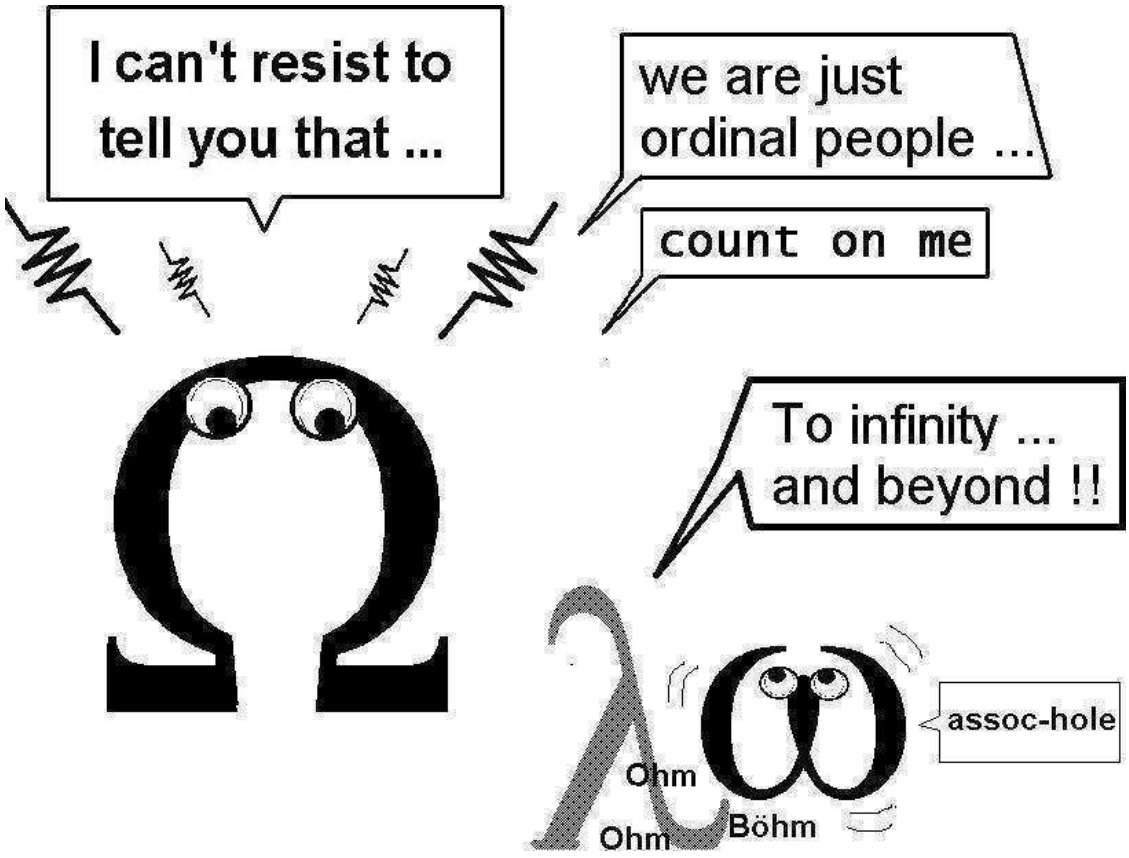
Presentamos una extensión del cálculo $\lambda(\eta)$ que incluye un constructor de casos primitivo que se propaga a través de las abstracciones como una sustitución lineal de cabeza antes de actuar sobre los constructores, y probamos que este modo de descomposición del apareamiento de patrones permite recuperar la expresividad del estilo de los patrones de ML. Se demuestra que este sistema satisface confluencia, usando una técnica de "dividir y conquistar" semi automática por la cual se determinan todos los pares de sub sistemas de este cálculo que conmutan (considerando todas las combinaciones de las nueve reglas de reducción). Finalmente, se prueba

un teorema de separación (débil) para todo el formalismo, usando una técnica de separación inspirada en la técnica *Böhm-out*.

Por último, como otra faceta de exploración de sub cálculos, analizamos los términos que satisfacen la propiedad de expandir a términos puros, para λv y λs. Probamos que estos conjuntos de términos son propios y no recursivos. Como aplicación, se prueba la imposibilidad de mapeos adecuados entre ciertos pares de cálculos.

Contents

List of Figures


I can't resist to tell you that ...
we are just ordinal people ...
count on me
To infinity ... and beyond !!
assoc-hole
Ohm
Ohm
Böhm

Chapter 1

Introduction and preliminaries

The mind takes a long time to forget what has taken a long time to learn. – Seneca

One has to take care not that the reader could understand, but his need to understand. – Quintilianus

As a Roller in the Ocean, Life is Motion. – ABBA, *Move On*

ABSTRACT We start this thesis with a thorough introduction to rewriting, λ-calculus and explicit substitution concepts, mainly known results which will be used in the subsequent chapters.

1.1 How to read this thesis

Here we briefly describe different ways of reading this thesis, according to the familiarity of the reader with the topics treated.

After the introduction and preliminaries (this chapter), the rest of the chapters are organized in such a way that they can be read separately, although their order seems to be the most appropriate for reading them from a didactic point of view.

The reader who is completely unfamiliar with any aspect of rewriting, should read section 1.3 and the subsequent sections of this chapter. The reader who is somehow familiar with λ-calculus, or has used it partially, may skip section 1.5, although it is recommended for him/her to read the most of this chapter since we mention here many important results which will be used afterwards. The reader who is familiar with λ-calculus, but not so with abstract rewriting, may read section 1.3 and 1.6. Last, the reader who is familiar with all the topics of this thesis should go directly to the next chapter, where some motivations are presented, or directly to the chapter of interest.

The reader should care that all subsequent chapters could be read independently one from each other. But the general notation as well as the preliminary results to be used are fixed in this chapter.

We now move into the introduction.

1.2 Introduction

Transformations govern our world. Heraclitus (circa 450 b.C.) said that the entire world is in process of change, and one "never enters the same river twice". According to his philosophy, all the world is in perpetual change, even when we are not aware of that such as at the particle level. We understand the importance of change and study rewriting theory, which is a formulation of change in computer science. Change is important in every aspect of computing, thus rewriting is an appropriate formulation of theoretical computer science.

There has been an extensive research activity in rewriting theory during the last decades. Rewriting has an important number of sub-areas: general rewriting, λ-calculus formulations, semantics, term rewriting, higher-order rewriting, residual theory, explicit substitution, and other.

The plan of the chapter is as follows. After a general introduction, we make a (biased, selected) introduction to rewriting concepts, classical λ-calculus, and calculi with explicit substitution as variants of the former.

This introduction is informal, then we will formalize definitions starting from section 1.3.

1.2.1 Rewriting and computing

Rewriting can be seen as a "formulation of computing". We are speaking of "formulation" instead of "paradigm", since we believe that rewriting reestablishes the essence of computing in another (related) manner. For instance, term rewriting has the virtue (or defect) that leaves two kinds of choices as non-deterministic, namely the rule selection and the redex selection. This two facets of non-determinism constitute the nature of term rewriting in general.

In the following chapters we attack different problems concerning rewriting systems and λ-calculi with explicit substitution. We have selected problems concerning systems and their subsystems, in particular problems about typing, confluence, normalization, expansion and restriction.

1.2.2 Calculi

What is a calculus? We can discuss this question from the starting history of mathematics. In particular, classical differential and integral calculus (both constituting infinitesimal calculus)

in some sense are good examples of what the concept of calculus is or should be. It should allow a way of calculating, i.e. rewriting total or partial elements by selecting adequate rules and sub-elements.

A calculus should have rewriting rules, such as the chain rule in differential calculus, or the integration by parts rule in integral calculus. A rule can be applied to a given problem instance, thus it should be general. Intuitively, a calculus is just (and as much as) a way of calculation – transformation, over a specific set of objects.

In our exploration in the following chapters, different calculi in several contexts are proposed and studied. Actually a calculus is a rewriting system where a specific term syntax is given along with specific rules as well as compatibility rules. It is not the case of abstract rewriting, but it is also the case of (context-sensitive) term rewriting systems "downwards" in the formalism hierarchy (to be introduced in chapter 2). Anyway, it is more than that, since the term set should be manageable, at least the equality should be decidable, for instance this implies that the terms should be of finite length or at least they should have a description by comprehension which obviously should be finite. As a matter of fact, differential and integral calculi deal with series, which are some kind of infinite objects or terms. In this case, there is always the need of a way of specifying a general term. Thus, an infinite sum (series) is always finitely described. There were proposals of infinitary λ-calculi (whose terms may have infinite length). In these formulations, descriptions by comprehension are generally handled. The concept of calculation can still go through even on these infinite objects, but the concept of normal form and normalization may change. Nevertheless in this thesis we are interested in finitary calculi.

Since rewriting systems are essentially non-deterministic, the interest in rewriting strategies arises. Intuitively, a strategy is a criterion which selects for each element a rewrite step over the existing ones. We can distinguish different kinds of strategies when studying rewriting systems, such as constricting, perpetual, zoom-in, normalizing, maximal, minimal, etc. (4; 14).

1.2.2.1 λ-calculus

Lambda calculus, for short λ-calculus (11; 12; 19; 20), is a paradigmatic example of rewriting theory, with interesting and representative properties that make itself a main object of formulation, study and discussion. Historically it was the first rewriting system and it has most of the complications and subtleties one can find in the whole theory, despite of its apparent simplicity.

The objects of study in the λ-calculus are the terms of the language, as well as the ideas of convertibility or equality between pairs of these terms. As one can see, the terms are remarkably simple objects, specially considering that they can be somehow considered and used as programs. Note that λ-calculus (as well as the term rewriting paradigm) assumes the existence of a countably infinite set of variables as a starting point, for the simple (but significative) reason that one often needs to take a new (unused) variable in order to avoid

variables clash. The need of taking new variables is not only theoretical but also practical, in particular during α-conversion - see below. This is going to change when one considers λ-calculus in a de Bruijn setting, where variables are replaced by indices. Remarkably, this formulation solves some problems but opens new ones, which motivate many of the results to follow in our work.

How to perform substitution: the β-conversion. This relation is the main motivation of λ-calculus and has inspired its creation in the 1930s. Yet at present β-conversion still preserves its essence and profound motivation, with the particularity that it is extremely simple to formulate but complex to study. It relies in the possibility of binding an argument within a given term, and there is a conversion relation which establishes what it does with the argument. The power of the (untyped) λ-calculus comes from the lack of distinction between functions and arguments in some sense. Thus, the term $\lambda x.M$ behaves like a function, as if M was parameterized by the variable x. Application in the λ-calculus is like functional application. A function can even be its own argument. If M represents a term in which the variable x occurs free (possibly more than once), then the β-reduction has the task of substituting those occurrences by the actual argument. Note that any term can be a parameter (even in typed calculi, there is some freedom on this[1]). So the λ-calculus does not have that restriction (anyway it can have "partial restrictions" when considering typable terms).

1.2.3 de Bruijn indices

The main motivation of de Bruijn calculi (27) is avoiding the need of α-conversion, and hence eliminating variable names from the syntax, thus resulting in a relatively easier "implementation" of the calculus in a practical sense. De Bruijn calculi emerge in parallel with the main λ-calculus variations, as a result of modifying the syntax directly and with this "simple" motivation. It is remarkable that this apparently slight change in the formulation has deep consequences and introduces more requirements to study.

The use of indices make it possible for the syntax to become more "free" (in the sense of a free algebra), since the classic formulation treats with variable names which could collapse when taking an appropriate quotient structure to the whole set of terms. When using indices, no quotient is needed, so this means an advantage. Many proofs by induction on the terms set would proceed with a less number of cases, without the need of considering variable clashes, thus proceeding straightforwardly. Nevertheless, as a consequence more work is required with the subsequent difficulty to understand the calculus objects, since these kinds of representations are less intuitive. Much of the difficulty in dealing with specific calculi are due to the use of indices.

[1] This is to be contrasted with the substitution method of the π-calculus of Milner, Parrow and Walker in which only "names" (as opposed to all π-terms) may be used in the operation.

1.2.4 Patterns

Special attention should be devoted to *patterns*. Patterns -in a general sense- possibly constitute the goal of mathematics. A mathematician is expected to find patterns in real life, and to develop theories from them. The use of patterns in rewriting has been investigated (18; 21; 22; 41; 42; 80). Their inspiration comes from declarative -mainly functional- languages which handle these kinds of constructs (40; 68; 70). The goal of studying the use of patterns within rewriting is to model and understand these languages better.

We focus on a special kind of patterns in chapter 7, to obtain an extension of $\lambda_{\beta\eta}$-calculus, having *case constructs*, which can be both adequate for modeling pattern matching, and to gain *separation*, a property which shows that the calculus is complete in the sense that it provides sufficiently many reduction rules to identify all observationally equivalent normalizable terms.

1.2.5 Translations

Another common tool is the use of translations, and we will be using them. What is a translation? It mainly consists of a function which takes some object of a given set and returns an object of another given set. A translation can link two universes, apparently unrelated, and according to its properties, the structure of one could be translated to the other.

We will use and study some translations from one calculus syntax to another, in chapters 5, 6 and 8. These translations allow to prove properties of one of them by transferring the properties from the other. This does not mean that it is the only way to prove them for the latter. But perhaps the one which is more simple and effortless.

Remark that good translations use to map a subsystem into another one, and they are considered satisfactory applications when computing in one system is translated as computing in the other one. When a good translation does not exist, we are interested in proving the non-existence. This is done in chapter 8.

The idea of translation is related with rewriting itself. In all our formulations of translations, we deal with instantly translating expressions to other expressions, but not in a rewriting -step by step- manner. We give an example of a translation as a rewriting system in appendix A.

1.2.6 Substitution and explicit substitution

Substitution is omnipresent in Computer Science: it is a fundamental building block for defining β-reduction in the λ-calculus (functional programming (40; 68; 70)), it plays a preponderate role in unification (logic programming (65; 71)), it blends into the formalization of parameter passing methods (imperative programming), etc. Researchers have learnt that dismissing substitution as a simple (simultaneous) replacement operation could possibly become a mistake. The prime witness of this fact is the prolific development of the so called *calculi of*

explicit substitution: substitution which was usually regarded as an atomic process operating at the meta-level (our language of discourse) has been promoted to the object-level (our language of study), it has become a new *operator* in our language, that is, part of the syntax. Since operational properties of substitution are studied in the object-level, unexpected behavior at the time of implementation is minimized. Also, a fine-grained control of substitution is made available, for instance we may delay substitutions in order to avoid unnecessary duplication of information, this resulting in a more efficient implementation of the corresponding calculus.

During the last fifteen years -and before, in a somehow *implicit* manner-, many lambda calculi with explicit substitutions have been proposed and studied in the literature (for references, see (47; 59)). Among all the known calculi, the pioneering $\lambda\sigma$ (1) has been historically the first one as well as a model for others to compare with. This calculus reflects in its choice of operators and rules the calculus of categorical combinators (cf. (24)). The main innovation of the $\lambda\sigma$-calculus is the division of terms in two sorts: the sort `term` and the sort `substitution`. $\lambda\upsilon$ (13) has the same style, but with less operators and rules, thus constitutes a good example to work with. λs (46) departs from this style of explicit substitutions in two ways. First, it keeps the classical and unique sort `term` of the λ-calculus. Second, it does not use some of the categorical operators, especially those which are not present in the λ-calculus. λs introduces two new sets of operators which reflect the substitution and updating that are only present in the meta-language of the λ-calculus. By doing so, the λs-calculus can be said to be closer to the λ-calculus from an intuitive point of view, rather than a categorical one.

One can easily recognize that the concept of substitution is itself omnipresent in many theories, formulations and ideas within Computer Science (11; 15). It can be said that the main point of explicit substitution is that this important concept has been captured, in the sense that its properties become relevant when implementing it. We understand that explicit substitution is appropriate when treating a calculus' syntax, in a non-trivial higher-order or first order approach. We may ask the question of why the idea of a substitution is so important, and what is behind explicit substitution. Which are their main concepts? Recalling Abelson and Sussman (2), there is nothing in Computer Science which seems simple enough, but results more complicated in theory and practice, than *substitution*. Substitutions appear in one way or another behind the different notions of computing, from computability to compilation, interpretation and translation, from the theory to the implementation. Thus, it should not be so straightforward to handle or at least to understand and use them in a convenient way.

In order to benefit from the idea of explicit substitution, much work has been done, and more remains. Some key properties are necessary, or at least desirable, such as simulation, soundness, preservation of strong normalization (PSN), subject reduction, expansion to pure terms, subject reduction, weak or strong normalization of the substitution calculus, etc. The syntax concerning explicit substitution extends the old one, in order to include the substitution

operator, but it has to be done cautiously. The language of substitutions is subtle and should be studied in order to find out the behavior of the calculus.

There are also explicit substitution formulations for higher-order rewriting (83) (*Expression Reduction Systems*, *Combinatory Reduction Systems* (54), *Calculi of Indexed Names and Named Indices* (79), and others) as well as for other formalisms (π-calculus for instance).

1.2.7 Typing

One of the key points in declarative languages as well as logic and specification paradigms is type theory. Type theory began in the early 20th century with the work of B. Russell with the goal of solving inconsistency problems of set theory. Actually type theory was one of the possible solutions (others were proposed and studied by Zermelo-Frankel and Von Neumann).

Modern type theory can be seen as an extension of this work direction. The goal was to classify different objects according to their nature, possibly according to their main properties. For us, terms could have different types and this is a way of classification for them.

The use of types in an abstract setting has also been proposed, for example (83), where abstract rewriting systems with typing are considered, and subject reduction and type reduction is defined. Thus typed terms always define a subsystem of the parent calculus, as well as terms having a specific type.

1.2.8 Normalization

Typing allows for example to recognize a term as being strongly normalizing. Also, terms with a different type result non-equivalent in a system with subject conversion, in particular, when subject reduction holds. These two important properties link types with rewriting. In this sense we can say that types are relevant to rewriting theories and paradigms.

Typing can assist in a proof of normalization, even strong normalization. Typable terms use (and need) to be strongly normalizing or weakly normalizing for a calculus to be appropriate or good enough, both in theory and practice. The interest in this result has led to the discovery of different formulations of λ-calculus, as well as different explicit substitution formulations. Typing can motivate relations from one language to another one, and subject reduction becomes an important property as we will see.

Now, another problem is how to reach a normal form when it exists. In λ-calculus and some variations, strategies and standardization techniques were proposed and studied for a long time, and for instance reducing leftmost-outermost redexes one can always reach the normal form when it exists. Thus in general determining the existence of a normal form is, although undecidable, a semi-decidable problem. This would not be the case in some ARSs when, for instance, no standardization algorithm may exist. Else, when such an algorithm is not known, it

could be the case that this decision problem remains unsolved. We could not even know a priori if it is semi-decidable or not. This happens because the ARS paradigm is very general, hence there is a strong justification that decomposing it in different sub-formulations is adequate.

We now move into the preliminaries.

1.3 Rewriting

Rewriting is a technique, process, theory, ... which allows to experience, work with and formalize *transformation*. There are different paradigms of rewriting, which will be described next.

1.3.1 Abstract rewriting

We recall basic definitions.

Definition 1.3.1 *An* Abstract Rewriting System *(ARS) is a pair* $(A, \to)$ *where* A *is a set and* $\to$ *a binary relation on* A*.*

So an ARS is a pair consisting of a set and a binary relation on it, without any a priori structure. When $a \to b$ we say that a *reduces* (in one step) to b, we call $a \to b$ a *reduction* (or a one-step *derivation*). A (possibly infinite) sequence of reductions $a_1 \to a_2 \to a_3 \to \dots$ is called a *derivation* starting from a. We call *length* of that derivation its number of reductions (0 or more, finite or infinite). Some ARSs may admit or not admit infinite-length derivations as we shall see in the following subsection.

Sometimes we denote with A the ARS $(A, \to)$ when the relation $\to$ is clear from the context.

For any relation $\to$ over a fixed set, we denote with $\overset{+}{\to}$ its transitive closure (i.e. the least transitive relation which includes $\to$), with $\twoheadrightarrow$ its reflexive-transitive closure (i.e. the least reflexive and transitive relation which includes $\to$), and with $=_\to$ its reflexive-symmetrical-transitive closure (i.e. the least equivalence relation which includes $\to$).

Definition 1.3.2 *Let* $(A, \to)$ *be an ARS. For every* $x \in A$ *we denote* $\to(x) = \{y \in A \mid x \to y\}$*, and* $\to^{-1}(x) = \{y \in A \mid y \to x\}$ *the sets of immediate successors and immediate predecessors of* x*, respectively.*

Define $n_p^{\to}(x) = |\to^{-1}(x)|$ *the number of immediate predecessors of* x *(i.e. the number of possible one-step derivations ending in* x*), and* $n_s^{\to}(x) = |\to(x)|$ *the number of immediate successors of* x *(i.e. the number of possible one-step derivations starting from* x*). Sometimes we may omit the supra-index* $^{\to}$ *if the rewriting relation is clear from the context and no ambiguity arises.*

Definition 1.3.3 *Given an ARSs $\mathcal{A} = (A, \to_A)$ and a subset $B \subseteq A$, we define the set of successors of B as $\mathcal{S}(B) = \{y \in A \mid$ there exists $x \in B$ such that $x \twoheadrightarrow y\}$. We write $\mathcal{S}(\{x\})$ simply as $\mathcal{S}(x)$.*

An ARS $(A, \to_A)$ is finitely branching *(FB) if every element has a finite number of direct successors, i.e. $n_s^{\to_A}(x)$ is finite for all $x \in A$.*

An ARS $(A, \to_A)$ is bounded (out-)degree *(BD) if the number $max(\{n_s^{\to_A}(x) \mid x \in A\})$ is finite, i.e. there is an upper bound on the number of immediate successors of every element.*

Remark that the relation $\twoheadrightarrow$ can be recovered from the function $\mathcal{S}$, which in general is not the case of the relation $\to$, in the sense that two different rewriting relations may lead to the same function $\mathcal{S}$.

Note that BD implies FB, but the converse is not true.

1.3.2 Normalization

In any ARS, an element a is a normal form iff there is no element b such that $a \to b$. An element b is called a *normal form* of an element a iff $a \twoheadrightarrow b$ and b is a normal form. An element a is said *normalizing*, or *weakly normalizing* (WN) if there exists a normal form of a. An element is called *infinitely derivating*, if there exists an infinite derivation $a \to a_1 \to a_2 \to \ldots$ starting from a. An element is called *strongly normalizing* (SN) if it does not admit an infinite derivation starting from it, i.e. if it is not infinitely derivating. For every ARS $\mathcal{A}$, let $WN_{\mathcal{A}}$ denote the set of WN elements of $\mathcal{A}$, let $SN_{\mathcal{A}}$ denote the set of SN elements of $\mathcal{A}$, and let $\infty_{\mathcal{A}}$ denote the set of non-SN elements of $\mathcal{A}$. If an element is SN then it is clearly WN, but the converse is not true.

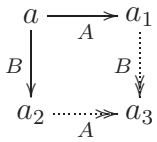
Normalization represents the halting of computations, i.e. the possibility of a process to stop. There are at least two important variations of normalization, due to the non-deterministic nature of rewriting: weak and strong normalization. Both of them represent halting possibilities in some manner.

1.3.3 Commutation and confluence

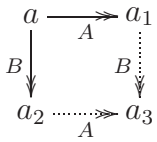
Confluence was presented by Church and Rosser (19; 20) and proved for the λ-calculus for the first time by them, and it resulted in a clue requirement for most rewriting systems afterwards.

Definition 1.3.4 *Given two relations $\to_A$ and $\to_B$ on the same set S, we say that $\to_A$* commutes weakly *with $\to_B$, and we denote it with $A//_w B$, if, for all $a, a_1, a_2 \in S$, if $a \to_A a_1$ and $a \to_B a_2$ then there exists $a_3 \in S$ such that $a_1 \underset{B}{\twoheadrightarrow} a_3$ and $a_2 \underset{A}{\twoheadrightarrow} a_3$. In other words, the*

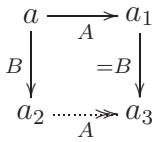
following diagram holds:



We say that $\to_A$ commutes *with* $\to_B$*, and we denote it with* $A//B$*, if, for all* $a, a_1, a_2 \in S$*, if* $a \underset{A}{\twoheadrightarrow} a_1$ *and* $a \underset{B}{\twoheadrightarrow} a_2$ *then there exists* $a_3 \in A$ *such that* $a_1 \underset{B}{\twoheadrightarrow} a_3$ *and* $a_2 \underset{A}{\twoheadrightarrow} a_3$*. In other words, the following diagram holds:*



We say that $\to_A$ commutes strongly *with* $\to_B$ *if, for all* $a, a_1, a_2 \in S$*, if* $a \to_A a_1$ *and* $a \to_B a_2$ *then there exists* $a_3 \in S$ *such that* $a_1 \to_{=B} a_3$ *and* $a_2 \underset{A}{\twoheadrightarrow} a_3$*. In other words, the following diagram holds:*



Note that strong commutation is not a symmetrical relation.

Definition 1.3.5 *We say that the ARS A (and also that the relation* $\to_A$*) is* weakly confluent*, or* weakly Church-Rosser *(WCR) if* $A//_w A$*. We say that the ARS A (and also that the relation* $\to_A$*) is* confluent*, or* Church-Rosser *(CR) if* $A//A$*. We say that the ARS A (and also that the relation* $\to_A$*) has the* diamond property *if for all* $a, a_1, a_2 \in S$*, if* $a \to_A a_1$ *and* $a \to_A a_2$ *then there exists* a_3 *such that* $a_1 \to_A a_3$ *and* $a_2 \to_A a_3$*.*

Actually, confluence can be formulated in other convenient equivalent ways.

Proposition 1.3.6 *Given any ARS, the following statements are equivalent:*

1. $\to$ *is confluent*
2. $\twoheadrightarrow$ *is WCR*
3. *If* $a_1 =_\to a_2$*, then there exists a term* a_3 *such that* $a_1 \twoheadrightarrow a_3$ *and* $a_2 \twoheadrightarrow a_3$ *(originally known as CR)*
4. *if* $a \to a_1$ *and* $a \twoheadrightarrow a_2$ *then there exists* a_3 *such that* $a_1 \twoheadrightarrow a_3$ *and* $a_2 \twoheadrightarrow a_3$ *(originally called* semi-confluence*).*

Of course, commutation (respectively confluence) implies weak commutation (resp. weak confluence), but the converse(s) do(es) not hold. Also, if the diamond property is satisfied, the system is confluent, but the converse does not hold.

When confluence holds, normal forms, when they exist, are unique, i.e. every term has at most one normal form.

Given the ARSs A and B, we denote with $A + B$ the (set theoretic) union of both relations.

Standard results we will use are the following two well-known lemmas.

Lemma 1.3.7 (Commutation Lemma) *If A commutes strongly with B, then $A//B$.*

PROOF: See (9) or (14). □

Lemma 1.3.8 (Newman's Lemma) *Let A be an ARS such that A is WCR and SN. Then, A is CR.*

PROOF: See (9), (14) or any other standard text on Rewriting theory. □

The importance of Newman's Lemma is that it connects two apparently unrelated notions: confluence and normalization. It will be extensively used, mainly on chapter 7.

A system which is both CR and SN is called *canonical*.

Confluence and weak confluence are key concepts of rewriting. The usual idea behind confluence is that from a single object, two derivations should eventually reach the same object. But there is another more profound idea behind this property. Confluence is in some manner a minimal property which should hold in order to be able to formulate an equational theory starting from a rewriting setting. That is, if confluence is guaranteed, then specific pairs of objects (for instance terms) would make sense as equations, otherwise some fundamental properties such as transitivity would cease to hold.

The definition of confluence takes three terms (one term and any two other successors). One could ask whether there could be another definition of confluence taking more than 2 derivations of a given term. A simple examination reveals that this notion would be equivalent to the classical with two reducts, i.e. stating the classical property implies this apparently generalized form with any finite number of successors in the hypothesis. This does not happen with weak confluence nor with the diamond property, even when the second implies confluence but the first does not.

1.3.4 Isomorphisms between systems

Two ARSs are *isomorphic* if there is a one-on-one mapping between one and the other which respects the one-step rewriting relation (in one way and in the other). More precisely, given the ARSs $\mathcal{A} = (A, \rightarrow_A)$, and $\mathcal{B} = (B, \rightarrow_B)$, an isomorphism between both is a bijection $f : A \rightarrow B$ such that $a \rightarrow_A a'$ iff $f(a) \rightarrow_B f(a')$. Isomorphic ARSs behave exactly in the same way as far as rewriting is concerned. Rewriting steps and derivations in one of them are translated into steps or derivations of the same characteristics in the other one.

1.3.5 Sub-ARSs

A *sub-ARS* of a given ARS is an ARS whose set of terms is a subset of the original set, closed under reduction, and with its relation being the restriction of the original ARS relation to the latter.

Definition 1.3.9 *Given two ARSs* $\mathcal{A} = (A, \rightarrow_A)$ *and* $\mathcal{B} = (B, \rightarrow_B)$ *we say that* $\mathcal{A}$ *is a* sub-ARS *of* $\mathcal{B}$ *if*

- $A \subseteq B$
- *the restriction of* $\rightarrow_B$ *to* A *is* $\rightarrow_A$*, i.e. for all* $x, y \in A$ $x \rightarrow_A y$ *iff* $x \rightarrow_B y$
- *for all* $x \in A, y \in B$*, if* $x \rightarrow_B y$*, then* $y \in A$*.*

Sub-ARSs always inherit the rewriting-related properties of the original ARS (such as being confluent, weakly confluent, normalizing, FB, etc.)

As a loosening of the definition of sub-ARS, a subsystem of a given rewriting system will be a subset of the elements with a relation which is a subset of the original one, which should be closed under reduction. The condition of being the restriction of the former is eliminated.

Definition 1.3.10 *Given two ARSs* $\mathcal{A} = (A, \rightarrow_A)$ *and* $\mathcal{B} = (B, \rightarrow_B)$ *we say that* $\mathcal{A}$ *is a* subsystem *of* $\mathcal{B}$ *if*

- $A \subseteq B$
- $\rightarrow_A \subseteq \rightarrow_B$

In the following sections we recall definitions of certain ARSs which have syntactical structure, in which their elements will be called *terms*.

1.4 First-order rewriting and string rewriting

This section briefly recalls rewriting in first-order signatures and in sets of strings.

1.4.1 Term rewriting

Term Rewriting Systems (TRSs) (9; 14) roughly consist of (context-free) rewriting over any first-order signature. A TRS consists of terms and rules for rewriting these terms.

A signature Σ consists of a non-empty set of *function symbols* or *operator symbols*, denoted $f, g, ...$, each equipped with a fixed *arity*, given by a natural number, which indicates the number of arguments it must have. Sometimes a symbol of arity n is called n-ary. An arity 1 symbol

is called unary, an arity 2 symbol is called binary, and a 0-arity symbol is called a constant, or nullary. An arity n symbol f is sometimes denoted with f^n in the examples.

Terms are strings of symbols taken from a so called *alphabet*, which is the signature plus a countably infinite set $\mathcal{X}$ of variables, denoted $x, y, z, ...$ which is disjoint from Σ.

The set of *terms over* Σ, denoted $T(\Sigma)$, is defined inductively as follows:

1. $x \in T(\Sigma)$ for every $x \in \mathcal{X}$.

2. if $n \geq 0$, f is an n-ary symbol and $t_1, \ldots, t_n \in T(\Sigma)$, then $f(t_1, \ldots, t_n) \in T(\Sigma)$ (it should be understood as just f if nullary).

The terms t_i are called the *arguments* of the term $f(t_1, \ldots, t_n)$, and the symbol f is the *head symbol* or the *root*. Terms not containing variables are called *ground* terms or *closed* terms, otherwise they are called *non-ground* terms, and in general all terms are called *open* terms (when they can possibly have variables). Terms in which no variable occurs more than once are *linear*. The length of a term t, denoted by $|t|$, is defined as expected:

$$\begin{array}{lcl} |x| & = & 1 \\ |f(t_1, \ldots, t_n)| & = & |t_1| + \cdots + |t_n| + 1 \end{array}$$

which measures the number of occurrences of function symbols plus variables.

A *context* is an "incomplete" term, a term with a unique *hole* denoted $\Box$, which is a distinguished element not in Σ. In a context, usually denoted $C\{\Box\}$, the replacement of the hole by a term t, denoted $C\{t\}$, is defined as expected.

The set of variables of a term t is defined as expected:

$$\begin{array}{lcl} Var(x) & = & \{x\} \\ Var(f(t_1, \ldots, t_n)) & = & Var(t_1) \cup \cdots \cup Var(t_n) \end{array}$$

A reduction rule for a signature Σ is a pair (l, r) of terms of $T(\Sigma)$ (note they may contain variables), to be written as $l \to r$, such that:

1. l is not a variable

2. $Var(r) \subseteq Var(l)$, that is, no variable which does not occur in l may occur in r

l is called the *left-hand side* (lhs) and r is called the *right-hand side* (rhs).

A substitution is a function $\sigma : \mathcal{X} \to T(\Sigma)$ such that $\sigma(f(t_1, \ldots, t_n)) = f(\sigma(t_1), \ldots, \sigma(t_n))$. Thus a substitution is always determined by its values in the set of variables $\mathcal{X}$. Other way to introduce a substitution is as a function from $\mathcal{X}$ to $T(\Sigma)$ and then extend it to the whole set of terms $T(\Sigma)$ such that the previous condition is fulfilled. For more details and results about this formulation of substitutions (with formulations of matching and unification problems), see (9; 14) (also any good text about logic programming).

A TRS is a pair (Σ, R) where R is a set of rules.

The rewriting relation of a TRS (Σ, R) is given as follows. Given two terms t and s, we say that t rewrites to s iff there exists a context C, a rule $l \to r \in R$ and a substitution σ such that $t = C\{\sigma(l)\}$ and $s = C\{\sigma(r)\}$.

In this case we say that $\sigma(l)$ is the *redex* and $\sigma(r)$ is the *contractum* or *reduct* of the redex.

A ground TRS is a TRS in which both lhs and rhs of every rule are ground terms.

Every TRS is FB. As a simplest example of a ground TRS which is not BD, take the one given by the unary symbol a, the binary symbol f and the rule: $a \to f(a, a)$.

1.4.2 String rewriting

There are different formulations of string rewriting, whether variables are included or not.

1.4.2.1 Semi-Thue Systems

Semi-Thue Systems (STSs) (9; 14) roughly consist of rewriting over strings, where the rules are pairs of strings. Given a non-empty finite set Σ of *symbols* or *characters*, called the *alphabet*, the set of *strings* over Σ, denoted Σ^*, consists of finite sequences of (zero or more) elements of Σ. If u and v are strings, their concatenation is denoted $u.v$ or simply uv.

A rewriting rule for Σ is a pair (l, r) of strings of Σ^*, written as $l \to r$. An STS is a pair (Σ, R) where R is a set of rules.

Given two strings t and s, we will say that t rewrites to s, written $t \to s$, if there exists a rule $l \to r$ and strings $u, v \in \Sigma^*$ such that $t = ulv$ and $s = urv$.

1.4.2.2 Post-Canonical Systems

Post Canonical Systems (PCSs) (14) basically consist of rewriting over strings without context compatibility, and allowing the use of variables in the rules.

A PCS is a rewriting system based on string rewriting, with a set of rules of the form $u \to v$, where u and v are strings over the alphabet $\Sigma \cup \mathcal{X}$, with Σ a non-empty finite set of symbols and $\mathcal{X}$ a denumerably infinite set of variables (notation $x, y, \dots$) disjoint from Σ, which will denote strings during the rewriting process, and with the (expected) restriction that every variable occurring in v must occur in u.

Substitutions can be defined in a way similar to substitutions for TRSs, with the difference that variables are mapped to strings from $(\Sigma \cup \mathcal{X})^*$. The empty string is denoted by ϵ. All substitutions are extended to handle arbitrary strings in an expected inductive way: for c any symbol, x any variable and v any string, $\sigma(\epsilon) = \epsilon$, $\sigma(cv) = c\sigma(v)$ and $\sigma(xv) = \sigma(x)\sigma(v)$.

The rewriting relation is defined as follows. Given two strings (possibly empty, possibly containing variables) t and s, we will say that t rewrites to s, written $t \to s$, if there exists a rule $l \to r$ and a substitution σ such that $t = \sigma(l)$ and $s = \sigma(r)$.

As it can be seen, there is an important restriction: rules do not manage to rewrite under the substring relation, that is, a rule completely specifies a rewriting step, where variables denote any substring and the lhs and rhs denote a string concatenation between their components (variables and/symbols). The rewriting relation is given as with TRSs but taking the context equal to a hole.

For that reason, a ground PCS (i.e. those without the use of variables) is not exactly a STS, since reduction steps do not necessarily occur in substrings matching a rule lhs. The rewriting relation defined by the PCS takes the entire string and rewrites it according to matching with some rule. But STSs are indeed a particular case of PCSs. For instance, to achieve the same effect than the STS rule $a \to bbc$, a PCS may have the rule $xay \to xbbcy$, having the variables x and y represent the string prefix and postfix respectively.

PCSs actually come from the logic arena, and they can be more general by allowing rules with a lhs having a (finite) set of strings instead of just one, thus stating that given a set of strings which have been deduced so far, a "rewriting" step consists in adding a new string to this set according to some rule. But we would not consider this general form since we are not dealing here with rewriting over sets nor any other structure which is not a first-order term or a string.

A final remark of this section. We have defined isomorphism between any pair of ARSs. Sometimes we will speak about (mixed) isomorphisms between an ARS and an STS, as well as between an ARS and a TRS, etc., making it clear that in every case we will mean *isomorphic as ARSs*. Remark that for example a TRS can have as a "graph" actually a *multi-graph*, i.e. some pairs of nodes may be connected by more than one edge. For instance, in the TRS with symbols $\{a^0, b^0, f^2\}$ and rules $f(a, x) \to x$ and $f(x, a) \to a$, the edge $(f(a, a), a)$ may "appear" twice. We will disregard this possibility when speaking about graph isomorphisms, since in our formulation ARS isomorphisms do not handle this. Thus when we take the graph of an ARS we implicitly collapse all edges connecting the same pair of nodes to a single one, everywhere where they may occur.

1.5 λ-calculus

λ-calculus will be a central issue in this thesis. Although we shall assume familiarity with classical λ-calculus with names (11), we briefly describe it here.

1.5.1 Classical λ-calculus

In the untyped λ-calculus (11) we should explain the role of the symbol λ as a binding operator. It has the goal of performing substitution in a given context. This notion is captured by equational theories over terms.

The set of λ-terms, denoted Λ, is described by the syntax:

$$M \quad ::= \quad x \mid (MM) \mid (\lambda x.M)$$

where x means "variables", i.e. x ranges over a given denumerably infinite set $\mathcal{X}$ of the so called variables, that is, elements that can be replaced during reduction as we shall see. Letters $M, N, O, P, \ldots$ shall be used for λ-terms. Free and bound variables are defined as follows. We denote with $FV(M)$ the set of free variables of M which is defined as

$$\begin{array}{lcl} FV(x) & = & \{x\} \\ FV(MN) & = & FV(M) \cup FV(N) \\ FV(\lambda x.M) & = & FV(M) \setminus \{x\} \end{array}$$

Bound variables of a term are defined as expected:

$$\begin{array}{lcl} BV(x) & = & \emptyset \\ BV(MN) & = & BV(M) \cup BV(N) \\ BV(\lambda x.M) & = & BV(M) \cup \{x\} \end{array}$$

Remark that a variable may appear bound and free in different occurrences.

A context $C\{\Box\}$ is a term with a unique hole, which can be replaced by any term, in the expected way.

1.5.1.1 α-conversion

We recall the first of two notions of convertibility (also called conversion) between terms which is described by a relation, namely α-conversion.

α-conversion is stated inductively as follows:

$$\begin{array}{llll} \lambda x.M & =_\alpha & \lambda y.M\{x \leftarrow y\} & \text{if } y \neq x, y \notin FV(M) \\ \text{if } M =_\alpha M' & \text{then} & MN =_\alpha M'N & \\ \text{if } N =_\alpha N' & \text{then} & MN =_\alpha MN' & \\ \text{if } M =_\alpha M' & \text{then} & \lambda x.M =_\alpha \lambda x.M' & \end{array}$$

The significance of this relation relies in the fact that variables are born all equal, and there is no need to make them differ one from each other when they get the status of bound variables. We can make sure that the bound variables of terms do not interfere with each other, nor with any free variables. For this we usually adopt the so called Barendregt's "variable convention", which can be vaguely stated: *In any definition, theorem or proof in which only finitely many terms appear, we silently α-convert them so that bound variables of each term are not the same*

as the bound variables of any other term, or the free variables of any term. This principle is of meta-level, thus it can be seen as higher-order with respect to normal contexts we use to manage within a mathematical theory.

Syntactical equality (modulo renaming of bound variables) is expressed using the $=$ symbol.

For every pair of terms M and N, M is a *sub-term* of N, denoted $M \subseteq N$, iff $M \in Sub(M)$ where the set of terms $Sub(M)$ is defined as follows:

$$\begin{array}{lcl} Sub(x) & = & \{x\} \\ Sub(MN) & = & Sub(M) \cup Sub(N) \cup \{MN\} \\ Sub(\lambda x.M) & = & Sub(M) \cup \{\lambda x.M\} \end{array}$$

1.5.1.2 β-conversion

We recall $M\{x \leftarrow N\}$ which is used to denote meta-level substitution of all free occurrences of x in M by N (see (11)). The formal definition is:

$$\begin{array}{lcll} x\{x \leftarrow P\} & = & P & \\ y\{x \leftarrow P\} & = & y & y \neq x \\ (MN)\{x \leftarrow P\} & = & (M\{x \leftarrow P\})(N\{x \leftarrow P\}) & \\ (\lambda y.M)\{x \leftarrow P\} & = & \lambda y.(M\{x \leftarrow P\}) & y \neq x \end{array}$$

Without the use of the free-variable convention, it would be necessary a clause like $(\lambda y.M)\{x \leftarrow P\} = \lambda u.(M\{y \leftarrow u\})\{x \leftarrow P\})$ for $y \neq x, u \notin FV(M) \cup FV(P)$ i.e. a fresh variable u is taken, for handling the case $y \in FV(P)$ (and avoiding the so called *variable clash* or *capture*).

Intuitively $\{x \leftarrow N\}$ is seen to traverse M until it reaches its variables and then either a copy of N is discarded or replaces the occurrence of x in question.

The β-reduction of λ-calculus is described next. We write $M \rightarrow_\beta N$ when M *reduces in one* β*-step* to N, that is $M = C[(\lambda x.P)Q]$ and $N = C[P\{x \leftarrow Q\}]$ where C is some context.

Thus the β-rewrite rule reads: $(\lambda x.M)N \rightarrow_\beta M\{x \leftarrow N\}$. Intuitively, $\lambda x.M$ denotes a one argument function whose (formal) parameter is named x, and N is the actual parameter to which this function is applied. The result of applying $\lambda x.M$ to N is a new term, denoted $M\{x \leftarrow N\}$, obtained from M by substituting all (free) occurrences of x by N. Note that this operation is executed atomically, in one go.

It is a fundamental theorem of Church and Rosser that λ-calculus is confluent (19; 20).

A reduction strategy $\rightarrow_s$ is complete with respect to a subset B of terms if, given any term a, if $a \twoheadrightarrow b$ with $b \in B$, then $a \underset{s}{\twoheadrightarrow} b$. For instance, it is known that in lambda calculus the *leftmost-outermost* reduction strategy is complete with respect to the set of normal forms.

1.5.1.3 η-conversion

The η-reduction, sometimes also called *extensionality*, is the rule

$$\lambda x.Mx \quad \to \quad M \quad \text{if } x \notin FV(M)$$

with its compatible closure. The intuition behind η-reduction and its associated equivalence relation, η-conversion (denoted $=_\eta$) is that two "functions" can be considered equal if yield identical results when applied to identical arguments.

$\lambda_{\beta\eta}$-calculus, which is λ-calculus with both β- and η-reduction, is also confluent.

1.5.2 λ-calculus à la de Bruijn

Although familiarity is assumed with de Bruijn notation (27) and meta-substitution, we now briefly describe this λ-calculus formulation.

As we said the motivation behind the use of indices instead of variables is to avoid the need of α-conversion. Natural numbers (indices) are used to represent the variables, according to the nesting of λ-binders.

In the context of the λ-calculus à la de Bruijn we denote with Λ_{dB} the set of de Bruijn terms, whose syntax is as follows:

Definition 1.5.1 *1. The syntax of the* λ-calculus *terms in de Bruijn notation is given by:*

$$a \quad ::= \quad n \mid (aa) \mid (\lambda a) \quad \textit{where } n \in \mathbb{N}.$$

2. We say that a reduction $\to$ *is* compatible on Λ_{dB} *when for all* a, b, $c \in \Lambda_{dB}$, *if* $a \to b$ *then* $a\,c \to b\,c$, $c\,a \to c\,b$ *and* $\lambda a \to \lambda b$.

3. β-reduction *is the smallest compatible reduction on* Λ *generated by:*

$$(\beta\textit{-rule}) \qquad (\lambda a)\,b \to_\beta a\{\!\{1 \leftarrow b\}\!\}$$

where $\bullet\{\!\{\bullet \leftarrow \bullet\}\!\}$ *is the usual meta-substitution operator for the de Bruijn terms (see below). The* λ-calculus (à la de Bruijn), *is the reduction system whose only rewriting rule is* β.

We recall the definition of free variables of de Bruijn terms as follows:

$$\begin{array}{lcl} FV(m) & = & \{m\} \\ FV(ab) & = & FV(a) \cup FV(b) \\ FV(\lambda a) & = & FV(a) - 1 \end{array}$$

where $A - n = \{m - n \mid m \in A, m > n\}$.

We also recall the updating functions $U_k^i(\bullet)$ for $i \geq 1$, $k \geq 0$ as follows:

$$
\begin{array}{lcl}
U_k^i(m) & = & \left\{ \begin{array}{lll} m+i-1 & \text{if} & m > k \\ m & \text{if} & m \leq k \end{array} \right. \\
U_k^i(ab) & = & U_k^i(a)U_k^i(b) \\
U_k^i(\lambda a) & = & \lambda U_{k+1}^i(a)
\end{array}
$$

Last, we recall the meta-substitution operator at level i, for $i \geq 1$, of a term $b \in \Lambda_{dB}$ in a term $a \in \Lambda_{dB}$, denoted $a\{\!\{\mathtt{i} \leftarrow b\}\!\}$, as follows:

$$
\begin{array}{lcl}
n\{\!\{\mathtt{i} \leftarrow b\}\!\} & = & \left\{ \begin{array}{lll} n-1 & \text{if} & n > i \\ U_0^i(b) & \text{if} & n = i \\ n & \text{if} & n < i \end{array} \right. \\
(cd)\{\!\{\mathtt{i} \leftarrow b\}\!\} & = & (c\{\!\{\mathtt{i} \leftarrow b\}\!\})(d\{\!\{\mathtt{i} \leftarrow b\}\!\}) \\
(\lambda c)\{\!\{\mathtt{i} \leftarrow b\}\!\} & = & \lambda(c\{\!\{\mathtt{i}+\mathtt{1} \leftarrow b\}\!\})
\end{array}
$$

There is a celebrated isomorphism between classical λ-calculus and the λ-calculus à la de Bruijn (43; 44), thus the latter is also confluent. This isomorphism is given by the pair of translation functions $w_{[x_1,\ldots,x_n]}(\bullet) : \Lambda \to \Lambda_{dB}$ and $u_{[x_1,\ldots,x_n]}(\bullet) : \Lambda_{dB} \to \Lambda$ defined below.

For every term $M \in \Lambda$ such that $FV(M) \subseteq \{x_1, \ldots, x_n\}$ we define, by induction on M, $w_{[x_1,\ldots,x_n]}$ as follows:

$$
\begin{array}{lcl}
w_{[x_1,\ldots,x_n]}(v) & = & min\{j \mid v = x_j\} \\
w_{[x_1,\ldots,x_n]}(MN) & = & w_{[x_1,\ldots,x_n]}(M)w_{[x_1,\ldots,x_n]}(N) \\
w_{[x_1,\ldots,x_n]}(\lambda x.M) & = & \lambda w_{[x,x_1,\ldots,x_n]}(M)
\end{array}
$$

Let $\mathcal{X} = \{v_1, v_2, \ldots\}$ the set of all variables, then for every term $M \in \Lambda$ we define $w(M) = w_{[v_1,v_2,\ldots,v_n]}(M)$ where n is such that $FV(M) \subseteq \{v_1, v_2, \ldots, v_n\}$.

For every term $a \in \Lambda_{dB}$ such that $FV(a) \subseteq \{1, \ldots, n\}$ we define, by induction on a, $u_{[x_1,\ldots,x_n]}$ as follows:

$$
\begin{array}{lcll}
u_{[x_1,\ldots,x_n]}(i) & = & x_i & \\
u_{[x_1,\ldots,x_n]}(bc) & = & u_{[x_1,\ldots,x_n]}(b)u_{[x_1,\ldots,x_n]}(c) & \\
u_{[x_1,\ldots,x_n]}(\lambda b) & = & \lambda x.u_{[x,x_1,\ldots,x_n]}(b) & \text{with } x \notin \{x_1, \ldots, x_n\}
\end{array}
$$

Let $\mathcal{X}$ be the same enumeration of variables as above, then for every $a \in \Lambda_{dB}$ we define $u(a) = u_{[v_1,v_2,\ldots,v_n]}(a)$ where n is such that $FV(a) \subseteq \{1, 2, \ldots, n\}$.

Both definitions of w and u are correct, i.e. do not depend on the n taken nor on the choice of x (see (44)). The pair (w, u) realizes the isomorphism between both calculi in the sense that for every $a \in \Lambda_{dB}$ $w(u(a)) = a$ and for every $M \in \Lambda$ $u(w(M)) =_\alpha M$. We will take α-conversion as equality from now onwards, thus we accept that w and u are inverses of each other.

1.5.3 Types and environments

We give a brief survey of the Curry-style simply typed versions of the λ-calculus with de Bruijn indices (47).

Types and environments (or contexts) are defined for most of the calculi discussed in this thesis as follows:

Definition 1.5.2 *The syntax for types and environments in the de Bruijn setting is given by:*

$$\textbf{Types} \quad \mathcal{T} ::= T \mid \mathcal{T} \to \mathcal{T} \qquad\qquad \textbf{Environments} \quad \mathcal{E} ::= \mathit{nil} \mid \mathcal{T}, \mathcal{E}$$

where T is a set of basic types. We let A, B, etc. range over $\mathcal{T}$ and E, F, etc. range over $\mathcal{E}$.

Sometimes we will refer to environments as *contexts*. The following notation for environments will be frequently used. For an environment $E = A_1, A_2, \ldots, A_n$, we denote with E_i the i-th type of the environment, i.e. A_i, with $E_{\geq i}$ the environment $A_i, A_{i+1}, \ldots, A_n$ and with $E_{>i}$ the environment $A_{i+1}, A_{i+2}, \ldots, A_n$. Similarly, $E_{\leq i}$ denotes the environment $A_1, A_2, \ldots, A_i$; $E_{<i}$ is the environment $A_1, A_2, \ldots, A_{i-1}$, and $E_{<i,B,\geq i}$ is the environment

$$A_1, A_2, \ldots, A_{i-1}, B, A_i, \ldots, A_n$$

i.e. the result of adding type B before position i to the environment E.

$$(\mathbf{L1} - \mathit{var}) \quad A, E \vdash \mathtt{1} : A \qquad\qquad (\mathbf{L1} - \lambda) \quad \frac{A, E \vdash b : B}{E \vdash \lambda b : A \to B}$$

$$(\mathbf{L1} - \mathit{varn}) \quad \frac{E \vdash \mathtt{n} : B}{A, E \vdash \mathtt{n+1} : B} \qquad\qquad (\mathbf{L1} - \mathit{app}) \quad \frac{E \vdash b : A \to B \qquad E \vdash a : A}{E \vdash b\, a : B}$$

Figure 1.1: The typing rules of the simply typed λ-calculus à la de Bruijn

The typing rules for the simply typed λ-calculus in de Bruijn notation are defined in Figure 1.1. We call this typing system **L1** (and it is going to be extended in short for other calculi). This typing system satisfies several good properties: subject reduction and strong normalization of typable terms (cf. (11)).

1.6 Explicit substitution

Classical λ-calculus has the meta-substitution operator which behaves as an atomic or primitive operation. Since there are many interesting questions regarding the way substitutions are

"executed", calculi of explicit substitution for the λ-calculus were created. The aim is to fill the gap between theory and implementation of the λ-calculus by making the substitution notion *explicit*, that is, part of the syntax of the calculus.

We give in the following subsections just a minimal briefing about $\lambda\mathtt{x}$, λv, λs, λs_e, $\lambda\omega$ and $\lambda\omega_e$ for understanding the motivations behind explicit substitution, as well as the main problems of this thesis to be discussed later on. For more details about these rewriting systems, see the references cited for each one of them.

1.6.1 The $\lambda\mathtt{x}$-calculus

The $\lambda\mathtt{x}$-calculus (15; 76) is a calculus of explicit substitutions for the λ-calculus formulated in a named variable setting. The operation of substitution is incorporated at the object-level, thus the β-reduction rule is transformed into the following *Beta*-reduction rule: $(\lambda x.M)N \rightarrow_{Beta} M\langle x := N\rangle$ where the operator $\bullet\langle\bullet := \bullet\rangle$ is a new operator in the calculus to make substitutions explicit. Thus rules describing its behavior must be introduced.

Definition 1.6.1 ($\lambda\mathtt{x}$-terms) *Given a denumerably infinite set of variables $\mathcal{X}$ the $\lambda\mathtt{x}$-terms, denoted $\Lambda\mathtt{x}$, are given by the following syntax:*

$$M \quad ::= \quad x \mid (\lambda x.M) \mid (MM) \mid M\langle x := M\rangle$$

where x ranges over $\mathcal{X}$ as usual.

So now a term is either a variable, an application of a term to another term (represented by juxtaposition, as before), an abstraction, or a term of the form $P\langle x := Q\rangle$ called a closure*.*

The $\bullet\langle\bullet := \bullet\rangle$ operator is called the substitution operator*. Terms without occurrences of substitution operators are called* pure terms*. The usual variable convention is maintained with the additional observation that the variable x in the term $M\langle x := N\rangle$ binds the free occurrences of x in M; M is called the target and N the body of the substitution. $FV(M)$ denotes the set of free variables of a term M, defined as usual, with $FV(M\langle x := N\rangle) = FV(M) \setminus \{x\} \cup FV(N)$. The rewriting rules of the $\lambda\mathtt{x}$-calculus are given in Figure 1.2.*

The $\lambda\mathtt{x}$-calculus is confluent. The $\lambda\mathtt{x}$-calculus without the *Beta*-rule is called the *substitution sub-calculus of* $\lambda\mathtt{x}$ and is denoted by $\mathtt{x}$. This sub-calculus is SN and confluent, and its normal forms are pure terms (15). Thus if $M \in \Lambda\mathtt{x}$ then we use $\mathtt{x}(M)$ to denote its unique $\mathtt{x}$-normal form.

A variant of the $\lambda\mathtt{x}$-calculus is the $\lambda\mathtt{x}^{-}$-calculus whose rules are those of the $\lambda\mathtt{x}$-calculus except for the *Gc*-rule (also called *garbage collection*) which is replaced by the more *restricted* rule $y\langle x := P\rangle \rightarrow_{Var2} y$ where $x \neq y$. Note that $\lambda\mathtt{x}$ is more general than $\lambda\mathtt{x}^{-}$ in the sense that $\rightarrow_{\lambda\mathtt{x}^{-}} \subset \rightarrow_{\lambda\mathtt{x}}$ but $\rightarrow_{\lambda\mathtt{x}} \not\subset \rightarrow_{\lambda\mathtt{x}^{-}}$.

(Beta)	$(\lambda x.M)\ \mathrm{N}$	$\to$	$M\langle x := N\rangle$	
(App)	$(MN)\langle x := P\rangle$	$\to$	$M\langle x := P\rangle\ N\langle x := P\rangle$	
(Lam)	$(\lambda y.M)\langle x := P\rangle$	$\to$	$\lambda y.M\langle x := P\rangle$	$x \neq y$
(Var)	$x\langle x := P\rangle$	$\to$	P	
(Gc)	$M\langle x := P\rangle$	$\to$	M	$x \notin FV(M)$

Figure 1.2: The rewriting rules of the $\lambda\mathtt{x}$-calculus

It is easy to see that each β-rewrite step may be "implemented" by one *Beta*-rewrite step followed by many $\mathtt{x}$-rewrite steps. In some cases there might be pending substitutions that may not need to be executed. For example in the derivation below there has been no need to compute the substitution $(yy)\langle y := N\rangle$ thus reducing computation time and unnecessary duplication of the term N:

$$\begin{array}{lll}
(\lambda y.(\lambda x.z)(yy))N & \to_{Beta} & ((\lambda x.z)(yy))\langle y := N\rangle \\
& \to_{App} & (\lambda x.z)\langle y := N\rangle(yy)\langle y := N\rangle \\
& \to_{Lam} & (\lambda x.z\langle y := N\rangle)(yy)\langle y := N\rangle \\
& \to_{Gc} & (\lambda x.z)(yy)\langle y := N\rangle \\
& \to_{Beta} & z\langle x := (yy)\langle y := N\rangle\rangle \\
& \to_{Gc} & z
\end{array}$$

When augmenting the lambda calculus with explicit substitutions and assuming M β-rewrites to N, a rich supply of alternative derivations are provided in order to go from M to N. This suggests that rewrite strategies for the lambda calculus are relevant in this new setting. For instance, some work in optimal rewrite strategies for (weak) calculi of explicit substitutions has been addressed in (63) and normalizing strategies in (67), see also (30).

An important property that $\lambda\mathtt{x}$ fulfills is the *Preservation of Strong Normalization* (PSN). It means that given a SN term of classical λ-calculus, it is still SN in $\lambda\mathtt{x}$. One proof of this fact can be found in (15).

1.6.2 The $\lambda\upsilon$-calculus

The $\lambda\upsilon$-calculus (13; 60) is a calculus of explicit substitutions for the λ-calculus formulated in a de Bruijn setting with minimal substitution operators.

Definition 1.6.2 *The following is the two-sorted syntax for the $\lambda\upsilon$-terms and substitutions (where $\mathbb{N}$ denotes the set of positive natural numbers):*

Terms	$a ::= n \mid (a\ a) \mid (\lambda a) \mid a[s] \quad$ *where* $n \in \mathbb{N}$
Substitutions	$s ::= a/ \mid \uparrow \mid \Uparrow (s)$

Let Λ_υ^t be the set of $\lambda\upsilon$-terms, and Λ_υ^s be the set of $\lambda\upsilon$-substitutions, defined above.

Terms without occurrences of substitution operators are called pure terms. *A term of the form $a[s]$ is called a* closure, *where the sub-term a is called its* head.

The rewriting rules of the $\lambda\upsilon$-calculus are given in Figure 1.3.

$(Beta)$	$(\lambda a)b$	$\rightarrow$	$a[b/]$
(App)	$(ab)[s]$	$\rightarrow$	$a[s]b[s]$
(Lam)	$(\lambda a)[s]$	$\rightarrow$	$\lambda a[\Uparrow (s)]$
$(Fvar)$	$1[a/]$	$\rightarrow$	a
$(Rvar)$	$(n+1)[a/]$	$\rightarrow$	n
$(FvarLift)$	$1[\Uparrow (s)]$	$\rightarrow$	1
$(RvarLift)$	$(n+1)[\Uparrow (s)]$	$\rightarrow$	$n[s][\uparrow]$
$(VarShift)$	$n[\uparrow]$	$\rightarrow$	$n+1$

Figure 1.3: The rewriting rules of the $\lambda\upsilon$-calculus

We use $\lambda\upsilon$ to denote this set of rules. *Compatibility* on Λ_υ^t is defined by the following rules for terms a, b, c and substitutions s, t: if $a \rightarrow_{\lambda\upsilon} b$ then $ac \rightarrow_{\lambda\upsilon} bc$, $ca \rightarrow_{\lambda\upsilon} cb$, $\lambda a \rightarrow_{\lambda\upsilon} \lambda b$, $a/ \rightarrow_{\lambda\upsilon} b/$ and $a[s] \rightarrow_{\lambda\upsilon} b[s]$, and if $s \rightarrow_{\lambda\upsilon} t$ then $\Uparrow (s) \rightarrow_{\lambda\upsilon} \Uparrow (t)$ and $a[s] \rightarrow_{\lambda\upsilon} a[t]$. The rules (App), (Lam), (Fvar), (Rvar), (FvarLift), (RvarLift) and (VarShift) conform the υ-calculus. $\lambda\upsilon$ is confluent, and υ is confluent and SN. We denote with $\upsilon(u)$ the unique υ-normal form of a term or substitution u.

A $\lambda\upsilon$-context is defined in the expected way as follows:

Definition 1.6.3 *A* context *in $\lambda\upsilon$ is a term containing a unique hole $\Box$, i.e. generated by the following four-sorted syntax:*

Term contexts	$C ::= \Box \mid (C\ a) \mid (a\ C) \mid (\lambda\ C) \mid C[s] \mid a[S]$
Substitution contexts	$S ::= C/ \mid \Uparrow (S)$

where the syntax of terms a (Λ_υ^t) and substitutions s (Λ_υ^s) are those of Definition 1.6.2.

We recall the usual notion of *sub-term* and *positions* (given as sequences over {0,1}) of sub-terms (or holes) inside a given term (or context), and we denote with $Pos(a)$ the set of positions of the term a. We recall the notion of *prefix* and *proper prefix* between positions (corresponding to the nesting of sub-terms), as usual both for terms and contexts.

A *term context* is a context where the hole is in a sub-term (not substitution) position. Otherwise, the context will be a *substitution context*.

Definition 1.6.4 *For $s \in \Lambda_\upsilon^s$ we define the following substitutions as indicated:*

$$\begin{array}{lcl} \Uparrow^0 (s) & = & s \\ \Uparrow^{i+1} (s) & = & \Uparrow (\Uparrow^i (s)) \quad i \geq 0 \end{array}$$

Remark 1.6.5 *Every substitution in Λ_υ^s has either the form $\Uparrow^k (a/)$ for some $a \in \Lambda_\upsilon^t$ and $k \geq 0$ or the form $\Uparrow^k (\uparrow)$ for some $k \geq 0$.*

Proof: By induction on the derivation of its membership in Λ_υ^s. □

Definition 1.6.6 *For all $a \in \Lambda_\upsilon$ we define the size of a in the expected way, written $|a|$. Note that by the two-sorted fashion of Λ_υ we are defining the size not only for terms but also for substitutions in a mutually recursive way. For terms:*

$$\begin{array}{lcl} |m| & = & 1 \\ |\lambda a| & = & |a| + 1 \\ |ab| & = & |a| + |b| + 1 \\ |a[s]| & = & |a| + |s| + 1 \end{array}$$

and for substitutions:

$$\begin{array}{lcl} |a/| & = & |a| + 1 \\ |\uparrow| & = & 1 \\ |\Uparrow (s)| & = & |s| + 1 \end{array}$$

$\lambda\upsilon$ satisfies PSN, with respect to the de Bruijn λ-calculus. For more details and proofs see (13; 60).

1.6.3 The λs-calculus

The λs-calculus (43) is a calculus of explicit substitutions for the λ-calculus formulated in a de Bruijn setting.

Definition 1.6.7 *1. The* set of terms, *noted* Λ_s, of the λs-calculus *is given as follows:*

$$a ::= n \mid (a\ a) \mid (\lambda a) \mid a\,\sigma^j a \mid \varphi_k^i a \qquad \textit{where} \quad j, i \geq 1, \quad k \geq 0, \quad n \geq 1.$$

A closure *is a term of the form $a\,\sigma^j b$. A* pure term *does not contain σ's nor φ's.*

2. The λs-calculus *is the rewriting system $(\Lambda_s, \rightarrow_{\lambda s})$, where $\rightarrow_{\lambda s}$ is the least compatible reduction on Λ_s generated by the rules in Figure 1.5.*

We use λs to denote this set of rules. *Compatibility* on Λ_s is defined by the following rules for terms a, b, c: if $a \rightarrow_{\lambda s} b$ then $ac \rightarrow_{\lambda s} bc$, $ca \rightarrow_{\lambda s} cb$, $\lambda a \rightarrow_{\lambda s} \lambda b$, $a\,\sigma^j c \rightarrow_{\lambda s} b\,\sigma^j c$, $c\,\sigma^j a \rightarrow_{\lambda s} c\,\sigma^j b$ and $\varphi_k^i a \rightarrow_{\lambda s} \varphi_k^i b$. The *s-calculus* is the rewriting system generated by the set of rules

$$(\mathbf{Ls1}-\sigma)\qquad \frac{E_{\geq i}\vdash b:B \qquad E_{<i},B,E_{\geq i}\vdash a:A}{E\vdash a\,\sigma^i b:A} \qquad (\mathbf{Ls1}-\varphi)\qquad \frac{E_{\leq k},E_{\geq k+i}\vdash a:A}{E\vdash \varphi_k^i a:A}$$

$$(\mathbf{Ls1}-Mtv)\qquad E\vdash X_{E,A}:A$$

Figure 1.4: Extra typing rules of the simply typed λs- and λs_e-calculi

$$\begin{array}{lrcl}
\sigma\text{-}generation & (\lambda a)\,b & \longrightarrow & a\,\sigma^1\,b \\
\sigma\text{-}\lambda\text{-}transition & (\lambda a)\,\sigma^j b & \longrightarrow & \lambda(a\sigma^{j+1}b) \\
\sigma\text{-}app\text{-}transition & (a_1\,a_2)\,\sigma^j b & \longrightarrow & (a_1\,\sigma^j b)\,(a_2\,\sigma^j b) \\
\sigma\text{-}destruction & \mathtt{n}\,\sigma^j b & \longrightarrow & \begin{cases} \mathtt{n-1} & \text{if} \quad n>j \\ \varphi_0^j\,b & \text{if} \quad n=j \\ \mathtt{n} & \text{if} \quad n<j \end{cases} \\
\varphi\text{-}\lambda\text{-}transition & \varphi_k^i(\lambda a) & \longrightarrow & \lambda(\varphi_{k+1}^i\,a) \\
\varphi\text{-}app\text{-}transition & \varphi_k^i(a_1\,a_2) & \longrightarrow & (\varphi_k^i\,a_1)\,(\varphi_k^i\,a_2) \\
\varphi\text{-}destruction & \varphi_k^i\,\mathtt{n} & \longrightarrow & \begin{cases} \mathtt{n+i-1} & \text{if} \quad n>k \\ \mathtt{n} & \text{if} \quad n\leq k \end{cases}
\end{array}$$

Figure 1.5: The rewriting rules of the λs-calculus

$s = \lambda s \setminus \{\sigma\text{-}generation\}$. λs is confluent, and s is confluent and SN. We denote with $s(a)$ the unique s-normal form of a term a.

λs also satisfies PSN, with respect to the de Bruijn λ-calculus. For more details and proofs about λs, see for example (15; 43; 74).

1.6.4 The λs_e-calculus

The λs_e-calculus has been proposed as an extension of λs in order to get the confluence on open terms for this calculus.

Definition 1.6.8 *1. The set of* open terms, *noted* Λs_{op} *is given as follows:*

$$a \ ::= \ X \mid n \mid (a\ a) \mid (\lambda\ a) \mid a\,\sigma^j a \mid \varphi_k^i\ a \quad \textit{where} \quad j, i \geq 1\,, \ \ k \geq 0\,, \ \ n \geq 1.$$

where X ranges over $\mathbf{V}$ which stands for a denumerably infinite set of variables (sometimes also called meta-variables*). Furthermore,* closures, pure terms *and* compatibility *are defined as for λs.*

2. λs_e is obtained by adding the rules of Figure 1.6 to those of Figure 1.5. The λs_e-calculus *is the reduction system $(\Lambda s_{op}, \rightarrow_{\lambda s_e})$ where $\rightarrow_{\lambda s_e}$ is the smallest compatible reduction on Λs_{op} generated by the set of rules λs_e. The* s_e-calculus *is the rewriting system generated by the set of rules $s_e = \lambda s_e \setminus \{\sigma\text{-}generation\}$.*

σ-σ-*tr*	$(a\,\sigma^i b)\,\sigma^j\,c$	$\longrightarrow$	$(a\,\sigma^{j+1}\,c)\ \sigma^i\,(b\,\sigma^{j-i+1}\,c)$	if	$i \leq j$
σ-φ-*tr 1*	$(\varphi_k^i\,a)\,\sigma^j\,b$	$\longrightarrow$	$\varphi_k^{i-1}\,a$	if	$k < j < k+i$
σ-φ-*tr 2*	$(\varphi_k^i\,a)\,\sigma^j\,b$	$\longrightarrow$	$\varphi_k^i(a\,\sigma^{j-i+1}\,b)$	if	$k+i \leq j$
φ-σ-*tr*	$\varphi_k^i(a\,\sigma^j\,b)$	$\longrightarrow$	$(\varphi_{k+1}^i\,a)\,\sigma^j\,(\varphi_{k+1-j}^i\,b)$	if	$j \leq k+1$
φ-φ-*tr 1*	$\varphi_k^i\,(\varphi_\ell^j\,a)$	$\longrightarrow$	$\varphi_\ell^j\,(\varphi_{k+1-j}^i\,a)$	if	$l+j \leq k$
φ-φ-*tr 2*	$\varphi_k^i\,(\varphi_\ell^j\,a)$	$\longrightarrow$	$\varphi_\ell^{j+i-1}\,a$	if	$l \leq k < l+j$

Figure 1.6: The new rewriting rules of the λs_e-calculus

Working with open terms in λs one loses confluence. This was solved in (45; 46) by introducing λs_e.

We recall the basic typing rules for λs and λs_e:

Definition 1.6.9 *The basic typing system* **Ls1** *is defined by adding to the rules of Figure 1.1, three new rules given in Figure 1.4. The first two rules are for both λs and λs_e. The third is only for λs_e. It is added to type open terms and should be understood as follows: for every meta-variable X, there exists an environment E and a type A such that the rule holds.*

Last, λs_e does not satisfy PSN.

1.6.5 The $\lambda\omega$- and $\lambda\omega_e$-calculi

In order to express λs-terms in the $\lambda\sigma$-style, (47) split the closure operator of $\lambda\sigma$ (denoted in a semi-infix notation as $-[-]$) into a family of closure operators that were denoted also with a semi-infix notation as $-[-]_i$, where i ranges over the set of natural numbers. (47) also admitted as basic operators the iterations of $\uparrow$ and therefore had a countable set of basic substitutions $\uparrow^n$, where n ranges over the set of natural numbers. By doing so, the updating operators of λs become available as $-[\uparrow^n]_i$. Finally, (47) introduced a *slash* operator of sort $\texttt{term} \to \texttt{substitution}$ which transforms a term a into a substitution $a/$. This operator may be considered as *consing with id* (in the $\lambda\sigma$-jargon, see subsection 1.6.6) and has been first exploited in the $\lambda\upsilon$-calculus (cf. (13)).

Definition 1.6.10 *1. The set of terms of the $\lambda\omega$-calculus, noted $\Lambda\omega$, is defined as $\Lambda\omega^t \cup \Lambda\omega^s$, where $\Lambda\omega^t$ (terms) and $\Lambda\omega^s$ (substitutions) are mutually defined as follows:*

Terms	a	$::= n \mid (a\ a) \mid (\lambda\ a) \mid a\ [s]_j$	*where $j \geq 1$, $n \geq 1$*
Substitutions	s	$::= \uparrow^i \mid a\ /$	*where $i \geq 0$*

The set $\lambda\omega$, of rules of the $\lambda\omega$-calculus *is given in Figure 1.7. The set of rules of the* ω-calculus *is $\lambda\omega \setminus \{\sigma\text{-}generation\}$.*

2. The set of open terms, *noted $\Lambda\omega_{op}$ is defined as $\Lambda\omega^t_{op} \cup \Lambda\omega^s_{op}$, where $\Lambda\omega^t_{op}$ (open terms) and $\Lambda\omega^s_{op}$ (open substitutions) are mutually defined as follows:*

Open Terms	a	$::= X \mid n \mid (a\ a) \mid (\lambda a) \mid a\ [s]_j$	*where $j \geq 1$*
Open Substitutions	s	$::= x \mid \uparrow^i \mid a\ /$	*where $i \geq 0$*

where X ranges over $\mathbf{V}$, a denumerably infinite set of term variables, and x ranges over $\mathbf{W}$, a denumerably infinite set of substitution variables.

3. The set of semi-open terms, *denoted $\Lambda\omega^t_{sop}$ is the set of open terms which do not contain substitution variables (i.e. only term variables are allowed).*

4. The set $\lambda\omega_e$, of rules of the $\lambda\omega_e$-calculus *is obtained by adding to the set of rules $\lambda\omega$ the set of rules given in Figure 1.8. The set of rules of the* ω_e-calculus *is $\lambda\omega_e \setminus \{\sigma\text{-}generation\}$.*

Remark that the schemes σ-σ-tr. and φ-σ-tr. of λs_e both translate into the same scheme of $\lambda\omega_e$, namely σ-/-trans.

$$
\begin{array}{lrcl}
\sigma\text{-}generation & (\lambda a)\, b & \longrightarrow & a\,[b/]_1 \\
\sigma\text{-}app\text{-}tr & (a\, b)[s]_j & \longrightarrow & (a\,[s]_j)\,(b\,[s]_j) \\
\sigma\text{-}\lambda\text{-}tr & (\lambda a)[s]_j & \longrightarrow & \lambda(a[s]_{j+1}) \\
\sigma\text{-}/\text{-}des & \mathtt{n}[a/]_j & \longrightarrow & \begin{cases} \mathtt{n-1} & \text{if } n > j \\ a[\uparrow^{j-1}]_1 & \text{if } n = j \\ \mathtt{n} & \text{if } n < j \end{cases} \\
\sigma\text{-}\uparrow\text{-}des & \mathtt{n}[\uparrow^i]_j & \longrightarrow & \begin{cases} \mathtt{n+i} & \text{if } n \geq j \\ \mathtt{n} & \text{if } n < j \end{cases}
\end{array}
$$

Figure 1.7: The rewriting rules of the $\lambda\omega$-calculus

$$
\begin{array}{lrcl}
\sigma\text{-}/\text{-}tr & a\,[b/]_k[s]_j & \longrightarrow & a\,[s]_{j+1}[b[s]_{j-k+1}/]_k \quad \text{if } k \leq j \\
/\text{-}\uparrow\text{-}tr & a\,[\uparrow^i]_k[b/]_j & \longrightarrow & \begin{cases} a[b/]_{j-i}[\uparrow^i]_k & \text{if } k+i \leq j \\ a[\uparrow^{i-1}]_k & \text{if } k \leq j < k+i \end{cases} \\
\uparrow\text{-}\uparrow\text{-}tr & a\,[\uparrow^i]_k[\uparrow^l]_j & \longrightarrow & \begin{cases} a[\uparrow^l]_{j-i}[\uparrow^i]_k & \text{if } k+i < j \\ a[\uparrow^{i+l}]_k & \text{if } k \leq j \leq k+i \end{cases}
\end{array}
$$

Figure 1.8: The new rewriting rules of the $\lambda\omega_e$-calculus

We describe now the basic typing rules for $\lambda\omega$ and $\lambda\omega_e$.

Definition 1.6.11 *The basic typing system of the $\lambda\omega$-calculus is called* $\mathbf{L}\omega\mathbf{1}$. *The rules are those given in Figure 1.1 together with the new rules given in Figure 1.9. As usual, rules (*$\mathbf{L}\omega\mathbf{1}$*-Mtv) and (*$\mathbf{L}\omega\mathbf{1}$*-Msv) are for open terms.*

$(\mathbf{L}\omega\mathbf{1} - id)$	$E \vdash \uparrow^0 \triangleright E$	$(\mathbf{L}\omega\mathbf{1} - slash)$	$\dfrac{E \vdash a : A}{E \vdash a/ : A, E}$
$(\mathbf{L}\omega\mathbf{1} - shift)$	$\dfrac{E \vdash \uparrow^i \triangleright E'}{A, E \vdash \uparrow^{i+1} \triangleright E'}$	$(\mathbf{L}\omega\mathbf{1} - clos)$	$\dfrac{E_{\geq j} \vdash s \triangleright E' \quad E_{<j}, E' \vdash a : A}{E \vdash a[s]_j : A}$
$(\mathbf{L}\omega\mathbf{1} - Mtv)$	$E \vdash X_{E,A} : A$	$(\mathbf{L}\omega\mathbf{1} - Msv)$	$E \vdash x_{E,E'} \triangleright E'$

Figure 1.9: Extra typing rules of the simply typed $\lambda\omega$- and $\lambda\omega_e$calculi

We recall the following results for the λs-, λs_e-, $\lambda\omega$ and $\lambda\omega_e$-calculi:

Theorem 1.6.12 (cf. (46; 47)) *The following hold:*

1. *The s- and ω-calculi are SN and confluent on Λs and $\Lambda\omega^t$ respectively.*
2. *Let $a, b \in \Lambda$ and $r \in \{\omega, s\}$. If $a \twoheadrightarrow_{\lambda r} b$ then $a \twoheadrightarrow_\beta b$. If $a \to_\beta b$ then $a \twoheadrightarrow_{\lambda r} b$.*
3. *The $\lambda\omega$- and λs-calculi are confluent on $\Lambda\omega^t$ and Λs respectively.*
4. *The $\lambda\omega$- and λs-calculi satisfy PSN*
5. *The ω_e- and s_e-calculi are WN and confluent.*
6. *The $\lambda\omega_e$- and λs_e-calculi are confluent on semi-open and open terms, respectively.*
7. *Let $a, b \in \Lambda$ and $r \in \{\omega_e, s_e\}$. If $a \twoheadrightarrow_{\lambda r} b$ then $a \twoheadrightarrow_\beta b$. If $a \to_\beta b$ then $a \twoheadrightarrow_{\lambda r} b$.*
8. **Subject Reduction of λr for $r \in \{s, \omega, s_e, \omega_e\}$:** *If $E \vdash a : A$ and $a \to_{\lambda r} b$ then $E \vdash b : A$, where the typing system should be understood as* **Ls1** *for $r \in \{s, s_e\}$ and as* $\mathbf{L}\omega\mathbf{1}$ *for $r \in \{\omega, \omega_e\}$.*
9. **SN of λr for $r \in \{s, \omega\}$:** *Every well typed term is SN in the simply typed λr-calculus.*

1.6.6 The $\lambda\sigma$-calculus

The $\lambda\sigma$-calculus (1; 25; 66) is historically the first calculus of explicit substitutions for the λ-calculus, originally based in *categorical combinators* (24). It was formulated in a de Bruijn setting, but the syntax has 1 as the only index.

Definition 1.6.13 *The following is the two-sorted syntax for the $\lambda\sigma$-terms and substitutions:*

$$\begin{array}{ll} \textbf{Terms} & a ::= 1 \mid (a\ a) \mid (\lambda a) \mid a[s] \\ \textbf{Substitutions} & s ::= id \mid \uparrow \mid (a \bullet s) \mid (s \circ s) \end{array}$$

Let Λ^t_σ be the set of $\lambda\sigma$-terms, and Λ^s_σ be the set of $\lambda\sigma$-substitutions, defined above.

A term of the form $a[s]$ is called a closure, *where the sub-term a is called its* head. *The $\circ$ operator is called* composition, *the $\bullet$ operator is called* cons *and the id operator is called* identity. *Note that there are no other indices than 1, thus indices are coded as follows: $1, 1[\uparrow], 1[\uparrow \circ \uparrow], 1[\uparrow \circ (\uparrow \circ \uparrow)], \ldots$, and then pure terms may include closures, compositions and $\uparrow$'s used in this way. Let $\uparrow^1 = \uparrow$, $\uparrow^{n+1} = \uparrow \circ \uparrow^n$ for $n \geq 1$. Then the following is the syntax for the $\lambda\sigma$-pure terms:*

$$\textbf{Pure terms} \quad a ::= 1 \mid 1[\uparrow^n] \mid (a\ a) \mid (\lambda a) \quad \textit{where } n \geq 1$$

The rewriting rules of the $\lambda\sigma$-calculus are given in Figure 1.10.

$(Beta)$	$(\lambda a)b$	$\rightarrow$	$a[b/]$
(App)	$(ab)[s]$	$\rightarrow$	$a[s]b[s]$
(Abs)	$(\lambda a)[s]$	$\rightarrow$	$\lambda a[1 \bullet (s \circ \uparrow)]$
$(Clos)$	$a[s][t]$	$\rightarrow$	$a[s \circ t]$
$(VarId)$	$1[id]$	$\rightarrow$	1
$(VarCons)$	$1[a \bullet s]$	$\rightarrow$	a
(IdL)	$id \circ s$	$\rightarrow$	s
$(ShiftId)$	$\uparrow \circ id$	$\rightarrow$	$\uparrow$
$(ShiftCons)$	$\uparrow \circ (a \bullet s)$	$\rightarrow$	s
(Map)	$(a \bullet s) \circ t$	$\rightarrow$	$a[t] \bullet (s \circ t)$
(Ass)	$(s \circ t) \circ u$	$\rightarrow$	$s \circ (t \circ u)$

Figure 1.10: The rewriting rules of the $\lambda\sigma$-calculus

$\lambda\sigma$ is confluent. The rules (App), (Abs), (Clos), (VarId), (VarCons), (IdL), (ShiftId), (ShiftCons), (Map) and (Ass) conform the σ-calculus, which is confluent and SN. We denote with $\sigma(u)$ the unique σ-normal form of a term or substitution u.

$\lambda\sigma$ does not satisfy PSN with respect to the de Bruijn λ-calculus (66). For more details and proofs see (15; 63; 66).

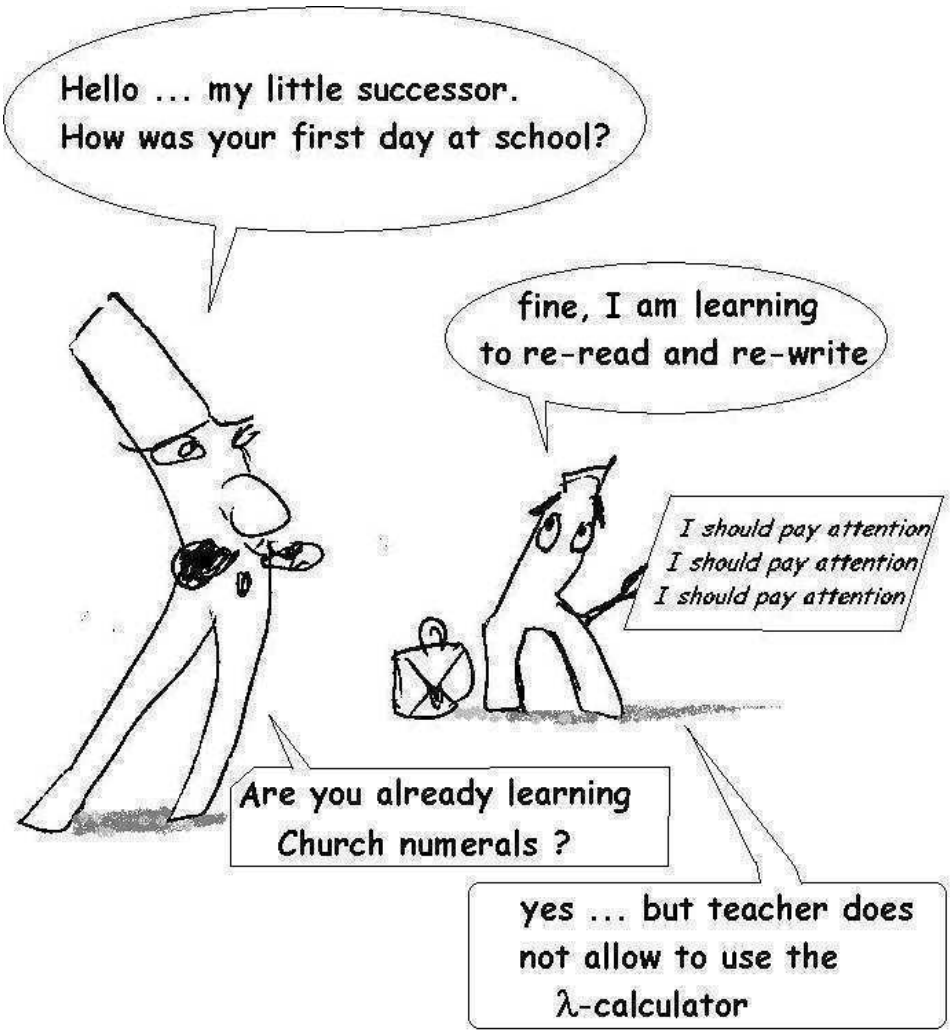
A final remark for this section. The PSN problem became interesting when it was observed that $\lambda\sigma$ did not satisfy this property, leading to infinite derivations which were not intended at the first glance when considering pure terms. Not only $\lambda\sigma$ lacks the PSN property. Another one is λs_e. Also $\lambda\mathtt{x}_{//}$ (15), which is similar to $\lambda\mathtt{x}$ but handles a *parallel* substitution operator. Last, another non-PSN calculus is $\lambda_\emptyset$ and its de Bruijn variations (see chapter 3).

1.7 General notation

Throughout the thesis, we use $\mathbb{N}$ to denote the set of (positive) natural numbers (1, 2, ...), and use the notation M, N, ... to range over terms of classical λ-calculus as well as over $\lambda\mathtt{x}$-terms, $a, b, \ldots$ to range over terms of any of the de Bruijn calculi treated, $s, t, \ldots$ to range over substitutions of any of the calculi having substitutions as a sort, and $m, n, \ldots$ to range over $\mathbb{N}$. In typed λ-calculi, we use $A, B, \ldots$ to denote types. Moreover, for terms as well as substitutions we denote with $u = v$ the fact that u and v are syntactically identical. We will assume from now onwards the usual conventions about parentheses (see (11; 12)), that is, when ambiguity does not arise we can omit parenthesis always taking applications, closures to be left associative and types to be right associative (for instance, $A \to B \to C$ means $A \to (B \to C)$). For every named λ-calculus we denote with $\lambda x_1 x_2 x_3 \ldots x_n.M$ the term $\lambda x_1.(\lambda x_2.(\lambda x_3.(\ldots(\lambda x_n.M))))$.

For every notion of reduction R we use the following standard notation:

- $\to_R$ for one R-reduction step, $\overset{+}{\to}_R$ for its transitive closure, and $\underset{R}{\twoheadrightarrow}$ or $\to_R^*$ for its reflexive transitive closure.
- $\to_R^n$ denotes the n-th composition of the relation $\to_R$ with itself (for $n \geq 0$), i.e. $u \to_R^n v$ iff there exist $u_0, u_1, \ldots, u_n$ where $u = u_0 \to_R u_1 \to_R \ldots \to_R u_{n-1} \to_R u_n = v$. If $n = 0$, $\to_R^n$ is the identity relation.
- If $u \underset{R}{\twoheadrightarrow} v$ we will also say that v R-*expands* (in zero or more steps) to u, and that u is an $(R-)$ expansion of v.


Hello ... my little successor.
How was your first day at school?
fine, I am learning
to re-read and re-write
I should pay attention
I should pay attention
I should pay attention
Are you already learning
Church numerals ?
yes ... but teacher does
not allow to use the
λ-calculator

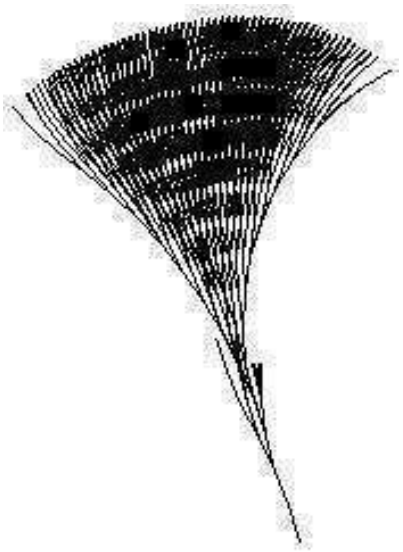


Figure 1.11: $LLL(LLLL)$ after 22 left-most steps

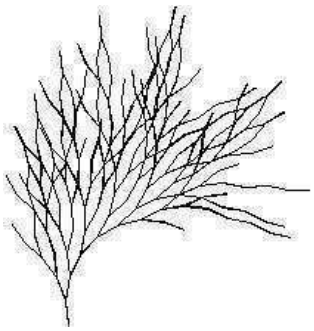


Figure 1.12: $S(S(SS)K)S(SSS)(S(S(SS))KS(SSS))$ after 24 left-most steps

Chapter 2

Motivating our work: systems, subsystems and relations

Science is organized knowledge. Wisdom is organized life. – I. Kant

Con el fin de buscar la verdad, es necesario, una vez a lo largo del curso de nuestra vida, dudar de todo tanto como sea posible. – R. Descartes

El primer paso hacia la filosofía es la incredulidad. – Anónimo

ABSTRACT We aim to study subsystems of general and specific abstract rewriting systems. We treat sub-ARSs of λ-calculus and other systems. We also exhibit a hierarchy of rewriting systems from the point of view of their reduction graphs. Given two rewriting paradigms, we are interested in knowing if any instance of one of them can be represented by some instance of the other one, and viceversa, in such a way that every single reduction step is preserved. We prove that this is not the case for the formalisms we study.

2.1 Introduction

Rewriting, not only from the philosophical but also from the mathematical and practical points of view, seems to be a powerful computing formulation, more precisely a family of paradigms.

This is a thesis about calculi with explicit substitution in the broad area of rewriting theory. The reader will access in the subsequent chapters to different problems about λ-calculi, mainly in the area of explicit substitution.

The idea of sub-structure turns out to be very interesting in most topics of rewriting theory, and that will be the main *invariant* of this thesis. In essence, we will treat several subsystems of λ-calculus. The concept of subsystem is thus very important, moreover its motivations

actually come from abstract algebra, category theory and logic. Important relations between rewriting systems come from considering subsystems and sub-ARSs as we will see. We also pose many examples of subsystems of given ARSs, many of which will be studied in the subsequent chapters.

Other question which comes to mind in the context of rewriting as computing is the generation of objects using rewriting rules, that is, the possibility of obtaining a given element from a set of elements (what in logic is usually called an inference). We can then formulate the concept of base, as a minimal set of generators for a given set, in the same way a base is a minimal set of generators in algebraic structures (such as a vector space, group, ring, algebra, etc.) This resembles the idea of a minimal set of axioms for a given theory.

In this chapter we are studying some rewriting formulations by looking at their complete reduction graphs. A natural question is to ask what is the relationship between rewriting theory and graph theory. The question seeks an explanation on what kind of problems rewriting can express, formulate or proof specifically, that graphs could not handle. At first glance every finite or infinite directed graph (even a multi-graph) can be seen as an ARS. And ARSs can be seen as directed graphs. But the main motivation of ARSs is completely different from the motivation of graph theory. In graphs, there is central interest in accessibility relation, and the interesting graphs use to be the finite ones. In ARSs, the central interest is normalization and confluence (as well as other properties which are essentially variations of the former), and most of the interesting ARSs are always infinite. In graphs, loops do not use to play an important role (sometimes they are simply excluded from analysis), while in ARSs a loop is just a particular case of cycle, which relates to non-normalization. Also, in graphs the existence of one or more paths for a pair of given nodes could be important, but it not always makes a difference if one or more than one exist. It is in general interesting to study properties of specific nodes, or pairs of nodes. Within ARSs, the existence of paths is crucial, although the formulation is different, and the main interest affects all objects and not a specific one, unless one considers sub-ARSs.

Perhaps one of the most significative differences is what concerns to decidability. In graph theory most interesting problems are decidable, even polynomial such as the existence of paths. Some of them though are *NP-complete*, as the existence of *hamiltonian cycles*, some kinds of sub-graphs, etc. In ARSs, since in general the relevant cases are infinite, most interesting problems are (far beyond the barrier of the) undecidable, like weak and strong normalization and confluence. Even restricting some formulations and problems for the λ-calculus, for some TRSs and for some STS, many of these problems may remain undecidable.

Given that rewriting paradigms are several and have very different presentations, in this chapter we propose a comparison between them. Our goal is to find important differences with respect to expressiveness, to get evidence of the necessity for the different rewriting formalisms, for identifying different classes of rewriting systems and understand a bit more of them. We

will consider the reduction graphs they have for comparing instances of one formalism with instances of another one. The main technique to apply will be to find different invariants which one formalism may have as a difference with others, as we will see.

Among existing rewriting paradigms, we focus on Abstract Reduction Systems (ARSs), Term Rewriting Systems (TRSs), Semi Thue Systems (STSs) and Post Canonical Systems (PCSs), as well as different forms of λ-calculus; and to investigate a number of relations between these formulations of rewriting.

2.2 Defining subsystems

There are three main ways of obtaining subsystems from rewriting systems:

1. restricting the set of elements
2. restricting set of rules (when they are given), and
3. restricting the replacement (when there is syntax), i.e. rewriting under specific positions (as in term or context-sensitive rewriting).

The first restriction always makes sense. The second one needs to have a calculus, where rules and a matching notion are defined. The third one needs to have a calculus and a term syntax. In this case we may call *sub-calculus* to a given subsystem. All these forms of restriction can be combined and many different subsystems can be found in this way. We will describe some examples in section 2.3.

Definition 2.2.1 *Given an ARS $\mathcal{A} = (A, \to_a)$, a sub-ARS $\mathcal{B} = (B, \to_b)$ of $\mathcal{A}$ and a subset $C \subseteq A$, we say that C is a generator of $\mathcal{B}$ if $\mathcal{S}(C) = B$.*

We will now characterize the sub-ARSs of any ARS in an abstract manner, with some immediate topological consequences, as a relation between generators and sub-ARSs.

Lemma 2.2.2 *For all ARSs $\mathcal{A} = (A, \to)$ one has the following (where $\mathcal{S}$ is given in Definition 1.3.3):*

1. $\mathcal{S}(\emptyset) = \emptyset$
2. *for $C \subseteq A$, $\mathcal{S}(\mathcal{S}(C)) = \mathcal{S}(C)$*
3. *for $C, D \subseteq A$, $\mathcal{S}(C \cup D) = \mathcal{S}(C) \cup \mathcal{S}(D)$ (and then also for finite unions)*
4. *moreover, for all families of sets $A_i \subseteq A$, $\mathcal{S}(\cup_i A_i) = \cup_i \mathcal{S}(A_i)$ (in particular, for all $B \subseteq A$, $\mathcal{S}(B) = \cup_{x \in B} \mathcal{S}(\{x\})$.*

PROOF: By definition using reflexivity and transitivity of $\twoheadrightarrow$. □

Actually any reflexive and transitive relation (here, $\twoheadrightarrow$) satisfies the previous result.

The first three items of Lemma 2.2.2 state that if we call the subsets C of A such that $\mathcal{S}(C) = C$ closed, then we have a *closure operator* S and therefore a topology in the set $\mathcal{P}(A)$ (56), in which the open sets are the complements of all closed sets. Closed sets are all sub-ARSs, as shown next.

2.2.1 Characterizing sub-ARSs

There is a clear relation between sub-ARSs and the $\mathcal{S}$ operator.

Proposition 2.2.3 *Let $\mathcal{A} = (A, \to)$ be an ARSs, and $B \subseteq A$. The following statements are equivalent:*

1. *$(B, \to |_B)$ is a sub-ARS of $\mathcal{A}$ (where $\to |_B$ is the restriction of the $\to$ relation to subset B and B is closed under $\to$)*
2. *there exists $C \subseteq A$ such that $B = \mathcal{S}(C)$*
3. *$B = \mathcal{S}(B)$*

PROOF:

- (3) $\Rightarrow$ (2) is obvious
- (2) $\Rightarrow$ (1) Let $x \in B = \mathcal{S}(C)$, such that $x \to y$, then $y \in \mathcal{S}(x) \subseteq \mathcal{S}(\mathcal{S}(C)) = \mathcal{S}(C) = B$
- (1) $\Rightarrow$ (3) $B \subseteq \mathcal{S}(B)$ is always true. To prove that $\mathcal{S}(B) \subseteq B$, let $y \in \mathcal{S}(B)$, thus there exists $x \in B$ such that $x \twoheadrightarrow y$, therefore $y \in B$, since closed under $\to$ implies closed under $\twoheadrightarrow$.

□

Therefore the sub-ARSs are the fixpoints of the $\mathcal{S}$ operator and also the closed sets of the topology.

2.3 Map of the territory: examples

We give here a first look to subsystems, actually, some general examples of sub-ARSs. Given an ARS A, the following are important sub-ARSs.

Example 2.3.1 *The set SN of strongly normalizing terms of A is a sub-ARS.*

Example 2.3.2 *The set of CR elements of A is a sub-ARS, where an element $a \in A$ is CR when the following implication holds: $a \twoheadrightarrow a_1, a \twoheadrightarrow a_2$ then there exists $a_3 \in A$ such that $a_1 \twoheadrightarrow a_3$ and $a_2 \twoheadrightarrow a_3$.*

Example 2.3.3 *The set WN of weakly normalizing terms of A is a sub-ARS when it is confluent.*

Example 2.3.4 *Given $k \in \mathbb{N}$, the set of terms a such that the set $\mathbb{S}(a)(= \{b \mid a \twoheadrightarrow b\})$ has cardinal not greater than k, is a sub-ARS.*

Example 2.3.5 *Given $k \in \mathbb{N}$, the set of globally finite terms a, is a sub-ARS, where a is globally finite if the set $\mathbb{S}(a) = \{b \mid a \twoheadrightarrow b\}$ has finite cardinal.*

Example 2.3.6 *Given an element b, the set $\{a \in A \mid a =_{\rightarrow} b\}$, the class of $\rightarrow$-equivalence of b, is a sub-ARS, where the relation $=_{\rightarrow}$ is the smallest equivalence relation including $\rightarrow$.*

Example 2.3.7 *Given a term N and $k \in \mathbb{N}$, the set $\{a \in A \mid maxred(a) \leq k\}$, the terms not admitting derivations longer than k, is a sub-ARS.*

TRSs have interesting examples of sub-ARSs too.

Example 2.3.8 *The set of ground terms of a TRS are a sub-ARS. In general, given a subset of variables V, the set of all terms having variables from the set V is a sub-ARS.*

Example 2.3.9 *If the TRS is orthogonal, the set of all elements which admit an infinite derivation, where in every step all erased subterms are SN ((14), ch. 4, sec. 4.8), is a sub-ARS.*

Now we move to λ-calculus.

For short, sub-ARSs of λ-calculus will be sometimes called *λ-sub-calculi*. In general it is difficult to identify and study all λ-sub-calculi, or to know how they look like. We will show examples next.

For appropriate typing systems we have the following

Example 2.3.10 *The set of well-typed terms, is a sub-ARS. This holds because of subject reduction.*

Example 2.3.11 *Let σ be a type and Γ a typing context. The set of terms M such that $\Gamma \vdash M : \sigma$, by subject reduction, is again a sub-ARS.*

Example 2.3.8 is also valid for λ-calculus. Let $Var(M)$ be the set of variables that occur in the term M. The following is also a family of λ-sub-calculi.

Example 2.3.12 *Let V' be a subset of V. Then the set $\Lambda_{V'} = \{M \in \Lambda \mid Var(M) \subseteq V'\}$ is a λ-sub-calculus.*

For this example to work, the Barendregt's free-variables convention must be used.

The previous examples can also be extended for most explicit substitution calculi, for instance $\lambda\mathbf{x}$. Also in λ-calculi à la De Bruijn similar examples of sub-ARSs can be given.

Remark 2.3.13 *The intersection of a family of sub-calculi is also a sub-calculus*

PROOF: It is clearly closed under reduction since all the members are. □

Remark 2.3.14 *There exist a non-denumerably infinite number of different λ-sub-calculi.*

PROOF: For each subset of variables $V_0 \subseteq V$ take the set $\{M \in \Lambda \mid FV(M) \subseteq V_0\}$ which is clearly closed under β-reduction, thus it is a sub-calculus. □

The same applies to TRSs. Anyway there exist more sub-ARSs than those generated in this manner.

The following example recalls context-sensitive rewriting (CSR) (61), in which a Σ-*map* (or *replacement map*) is joined to the signature indicating, for each function symbol, the set of its arguments under which reductions can be performed. If every Σ-map is trivial (i.e. the set of all arguments), we have a TRS as usual.

Example 2.3.15 *(61; 62) Given two* Context-Sensitive Rewriting Systems *(CSRSs)* (T_1, μ_1) *and* (T_2, μ_2) *with the same signature, and two Σ-maps μ_1 and μ_2 such that for each symbol f, $\mu_1(f) \subseteq \mu_2(f)$. Then (T_1, μ_1) is a subsystem of (T_2, μ_2) (not necessarily a sub-ARS). In particular, every CSRS is a subsystem (not necessarily a sub-ARS) of the corresponding TRS.*

The CSR paradigm via replacement maps can be extended to higher-order rewriting, having as an example λ-calculus itself. The inclusion or exclusion of rules μ, ν and ξ in the definition of β-reduction (see (11)) is the way of defining a replacement map, which specifies if it is possible to reduce under a specific symbol (or binder). λ-calculus without some of these rules has been studied, for instance *lazy λ-calculus* considers reduction without the ν-rule for modeling lazy functional programming.

Example 2.3.16 *Recall the λ_I-calculus (11) whose terms are given by the grammar*

$$M \quad ::= \quad x \mid MM \mid \lambda x.M \text{ where } x \in FV(M)$$

The last clause states that a term should always include all its abstracted variables. The β-reduction is the same as in classical λ-calculus, restricted for this syntax. An important property for this restricted version of λ-calculus is that a term is WN iff it is SN.

λ_I is another example of sub-ARS of classical λ-calculus, since the property that every term should include all its abstracted variables is preserved by reduction. And something analogous occurs if one considers λ_I with η-reduction.

Example 2.3.17 *In all studied calculi with explicit substitution, the associated substitution calculus is indeed a sub-calculus of the full calculus, because its rules are a subset of the full set of rules.*

Example 2.3.18 *A calculus of explicit substitution is always a proper sub-ARS of its formulation with open terms (also, semi-open terms), that is, the inclusion of term meta-variables in its syntax.*

Example 2.3.19 *In λ-calculus, the set of so called* quasi normal forms*, which are terms of the form $\lambda x_1 x_2 \dots x_n . x N_1 N_2 \dots N_m$ where $N_1, \dots, N_m$ are arbitrary terms, is a sub-ARS.*

Example 2.3.20 *In chapter 7 we will introduce a conservative extension of λ-calculus, which handles* case bindings*, that is the addition of case constructs to λ-calculus for modeling pattern matching. In this rewriting system we will work with a notion of* defined *and* undefined *terms. Defined terms can only reduce to defined terms, and undefined terms can only reduce to undefined terms. We will see that the set of* defined quasi-normal forms *is a sub-ARS, and so is the set of* undefined quasi-normal forms*. A remarkable feature is that we will have a partition of quasi normal forms into disjoint sub-ARSs as will be given by a Separation Theorem.*

2.4 Bases as good generators

When a subsystem of a system has been found, an appropriate way of describing it is to show how its elements are obtained, for example how can the system be generated. For this we may give a *generator set*. Better yet if that generator is minimal, or non-redundant. Therefore we propose the following Definition.

Definition 2.4.1 *Given an ARS $\mathcal{A} = (A, \to_a)$*

1. *An ARS-basis, $\to$-basis or simply a basis, of $\mathcal{A}$ is a minimal generator (in the sense of $\subseteq$).*
2. *An $=$-basis of $\mathcal{A}$ is a set D such that for all $M \in A$, there exists a unique $N \in D$ such that $N =_{\mathcal{A}} M$.*
3. *An $\leftarrow$-basis of $\mathcal{A}$ is a basis with respect to $\leftarrow$, the inverse relation of $\to$, i.e. a minimal set D such that for all $M \in A$, there exists $N \in D$ such that $M \underset{\mathcal{A}}{\twoheadrightarrow} N$.*

Since there are different ways of generating the existing subsystems, we have a reasonable notion of basis, that is a minimal set of terms which can be rewritten into any term of another given set of terms. The importance of a basis is that with a minimal number of elements one can characterize a given sub ARS so it can be generated in a "minimal" (non redundant) manner.

A little insight reveals that for having an $\rightarrow$-basis one would need exactly one term for each $\twoheadrightarrow$-branch, which can be in general a difficult thing to ensure.

It is in general undecidable to know whether some given set of terms is a basis (a $\leftarrow$-basis, an $=$-basis) of a given TRS. Actually, a much simpler problem, the word problem, is undecidable (9).

We investigate here the following question: does there exists a basis for λ-calculus? In that case, how to show one? Which other well known rewriting systems, such as term rewriting systems admit bases? We will prove that λ-calculi do not admit them, nor most of their "interesting" subsets of terms admit them.

Note that if we fix an $N \in \Lambda$ and let $\mathrm{B} = \Lambda - \{M \in \Lambda \mid M =_\beta N\}$, then B is not a basis, since $\mathcal{S}(B)$ will not contain N, otherwise there would be an $M \in B$ such that $M \underset{\beta}{\twoheadrightarrow} N$. On the other hand, Λ itself is clearly not a basis. So the problem is to find one or to prove non-existence of them. Remark that if one tries to prove the existence of a basis by using Zorn's Lemma, the proof fails. We will show that in fact this failure is essential, for there does not exist a basis of λ-calculus. To do it, we first give the following key property.

Lemma 2.4.2 *Let $\mathcal{B}$ be a basis of an ARS. Then for all $M \in \mathcal{B}$, N any term, if $N \twoheadrightarrow M$ then $M \twoheadrightarrow N$.*

Proof: If $N = M$ then $M \twoheadrightarrow N$ holds trivially. So we may suppose that $N \neq M$. Let $N \twoheadrightarrow M$. Then $N \notin \mathcal{B}$, or else by minimality $\mathcal{B}$ would not be a basis, and there exists $P \in \mathcal{B}$ such that $P \xrightarrow{+} N$. Suppose $P \neq M$, then $P \xrightarrow{+} M$, which is absurd since $\mathcal{B}$ is a basis. Then $P = M$, thus $M \xrightarrow{+} N$. □

Remark that the last lemma can be stated in an equivalent way as: for $\mathcal{B}$ a basis of an ARS, $M \in \mathcal{B}$ and N any term, if $N \xrightarrow{+} M$ then $M \twoheadrightarrow N$.

Now we can prove

Lemma 2.4.3 *The set of all λ-terms does not have a basis.*

Proof: Let $\mathcal{B}$ be a basis of λ-calculus. Let $M \in \mathcal{B}$, and let $u \notin FV(M)$, and let $x_1, x_2, \ldots$ be all variables. Note that for all $i \geq 1$, $(\lambda u.M)x_i \rightarrow_\beta M \in \mathcal{B}$. By Lemma 2.4.2 applied to λ-calculus, $M \xrightarrow{+}_\beta (\lambda u.M)x_i$, then $FV((\lambda u.M)x_i) \subseteq FV(M)$, so $x_i \in FV(M)$ for all $i \geq 1$, which is clearly absurd. Therefore bases do not exist. □

This lemma is interesting in itself, stating that the set of λ-calculus terms cannot be generated from the left in a "minimal" way.

We also have

Lemma 2.4.4 *The set of all λ-terms does not have a $\leftarrow$-basis.*

PROOF: If $\mathcal{B}$ is an $\longleftarrow$-basis of λ-calculus, take for example $T = (\lambda x.xxx)$ and $M = TT$, thus there exists a unique $N \in B$ such that $M \underset{\beta}{\twoheadrightarrow} N$. Since clearly $\mathcal{S}(T^i) = \{T^j |\ j \geq i\}$ then again taking $M' = T^{i+1}$ there should exist a unique $j \geq 2$ such that $M' \underset{\beta}{\twoheadrightarrow} T^j$, but then $M \underset{\beta}{\twoheadrightarrow} T^i \overset{+}{\rightarrow}_\beta T^j$, which contradicts the minimality of B. Therefore there is no such a set. □

As a consequence, we have that for all $C \subset \Lambda$, if $\mathcal{S}(C) = \Lambda$, then there exist $M, N \in C$ such that $M \neq N$ and $M \underset{\beta}{\twoheadrightarrow} N$. This is immediate by Definition 2.4.1 and Lemma 2.4.3. In other words, a set of generators of λ-calculus will necessarily have some redundance.

Corollary 2.4.5 *The set Λ_{dB} of all de Bruijn terms in λ-calculus à la de Bruijn does not have a basis nor an $\longleftarrow$-basis either.*

PROOF: One way is using Lemma 2.4.3 and the isomorphism between the de Bruijn and the classical formulations. Another proof is using the notion of free variable in the de Bruijn setting and a similar argument. □

Moreover, the question whether λ-calculus has an $=$-basis is immediately answered in the affirmative using the axiom of choice, by selecting a representative for each equivalence class. This holds for every ARS.

We could also repeat the question and obtain the same answer for most λ-calculus variations, such as explicit substitution calculi, and also for specific subsystems over those calculi. But in chapter 8 we will see that, for instance, Λ is not even a generator of the entire set of terms of some explicit substitution calculi.

Remark that in λ_I calculus (see Example 2.3.16) all abstractions should include the abstracted variable, thus the argument in Lemma 2.4.3 is no longer valid, since when $u \notin FV(M)$, $\lambda u.M$ is not a valid λ_I term, and this was used in the proof of Lemma 2.4.3.

We also have the following remarks concerning the existence or non-existence of bases and $\longleftarrow$-bases.

Remark 2.4.6 *The set NF of normal forms in any rewriting system has a basis (itself) which is also a $\longleftarrow$-basis.*

Remark 2.4.7 *The set SN of strongly normalizing terms in λ-calculus does not have a basis but it has a $\longleftarrow$-basis. Take the set of normal forms.*

Remark 2.4.8 *The set WN of weakly normalizing terms in λ-calculus does not have a basis but it has a $\longleftarrow$-basis. Take again the set of normal forms.*

2.5 Relationships between rewriting paradigms

We now proceed with the comparisons between the rewriting paradigms mentioned so far. Recall that a ground TRS is a TRS where all rules have ground terms as lhs and rhs. In other words, it is a TRS where no variables are present in the rules.

When comparing one TRS with another, sometimes one looks at their entire reduction graph and sometimes the reduction graph of their ground terms.

To start, it is known -and easy to show- that every STS is isomorphic to the restriction to ground terms of a linear TRS with only unary symbols and a constant (14). Also, remark that every ground TRS is isomorphic to a subsystem of an STS (perhaps not to the full STS), by coding the first order terms and rules with appropriate strings in an straightforward way including auxiliary symbols such as *comma* and *parenthesis*.

In this section we will show strict inclusions of classes of rewriting systems involving STSs, TRSs, PCSs and ARSs, and sometimes these relations involve the identification of a system with a subsystem (possibly a sub-ARS) of the other one.

From now onwards in the present chapter, unless explicitly stated, all TRSs, STSs, etc. are considered finitary, i.e. with a finite signature and a finite number of rules.

2.5.1 Main relations

We show and discuss the main relations among formalisms.

Remark 2.5.1 *If $(A, \to_A)$ and $(B, \to_B)$ are isomorphic ARSs, then the functions $n_p^{\to_A}$ and $n_p^{\to_B}$ coincide, and so do the functions $n_s^{\to_A}$ and $n_s^{\to_B}$.*

PROOF: Straightforward. □

The converse is not true, as it happens in graph theory: take on one side a circuit of $n \geq 6$ elements, and on the other two disjoint circuits of $n-k$ and k elements respectively, with $3 \leq k \leq n-3$. Both sides will not be isomorphic.

Lemma 2.5.2 *In every (sub-ARS of an) STS, for every string t, $n_p(t)$ and $n_s(t)$ are finite.*

PROOF: We prove the assertion for n_s. Since each rule consists of a pair of strings, then given any string x there is just a finite number of substrings, then a finite number of redexes (given that there are finite rules), thus the number of possible successors is finite.

The case of n_p is analogous since one can take the inverse STS, i.e. that in which every rule is the converse of a rule of the original STS, thus the role of successor and predecessors are symmetrical.

Finally, the property still holds when considering arbitrary sub-ARSs since in any sub-ARS both quantities never increase. □

The following Lemma states that n_p and n_s are not symmetrical.

Lemma 2.5.3 *There exist (linear) TRSs where, for some term t, $n_p(t)$ is infinite.*

PROOF: Take the TRS given by the signature $\Sigma = \{a^0, f^2\}$ and the rule $f(x, y) \to x$.

Then taking the term $t = a$, one has that $\to^{-1}(a)$ includes the set
$\{f(a, a), f(a, f(a, a)), f(a, f(a, f(a, a))), ...\}$
thus $n_p(a) = \infty$. □

We prove now that there are TRSs (with other good properties) which are not isomorphic to any sub-ARS of any STS. Proof is as follows.

Corollary 2.5.4 *There exist (linear and canonical) TRSs which are not isomorphic to (any sub-ARS of) any STS.*

PROOF: Take the above TRS which is canonical and use Remark 2.5.1 and Lemmas 2.5.2 and 2.5.3, then there is no STS nor sub-ARS of an STS which is isomorphic to this TRS. □

Nevertheless, every TRS is "almost" isomorphic to a sub-ARS of an augmented STS where rules can have variables denoting substrings (see subsection 2.5.3 below).

Now, the question: is every ARS isomorphic to some TRS? is answered negatively by an almost trivial syntactical argument in the following

Remark 2.5.5 *There are ARSs which are not isomorphic to any TRS.*

PROOF: Recall that every TRS is FB (since a term can only have finite redexes because of its finite length), and there are ARSs which do not fulfill this condition, so we are done. □

Other way to prove the assertion is using a cardinality argument. Given any subset $S \subseteq I\!N$, let $A_S = (I\!N \cup \{*\}, \to)$ be the ARS where $\to$ is defined by: $x \to y$ iff $y = x + 1$ or $x \in S$ and $y = *$. Then for every pair of sets $S, S' \subseteq I\!N$ such that $S \neq S'$, A_S is clearly not equal (even not isomorphic) to $A_{S'}$. This shows that there exist a non-denumerable number of ARSs, when the number of TRSs (with finite rules) is denumerable.

As a side note, consider multi-sorted signatures (9). Every n-sorted TRS is isomorphic to a subsystem (not necessarily a sub-ARS) of a one-sorted TRS which uses only one constant and one binary symbol. The idea of the proof relies on coding and *currification* for representing symbols of higher-arities.

2.5.2 Finite branching

The previous question subsists, now in the following form: is every ARS, satisfying FB and/or BD, isomorphic to some TRS? The answer is still negative, as we shall see in the next corollary (no matter if one is restricted to ground TRSs or not). We distinguish two kinds of reduction graphs: total (which includes all terms) and ground (which includes only the ground terms).

Proposition 2.5.6 *If the graph of all ground terms of a TRS is infinite, then it is isomorphic to some proper sub-graph.*

PROOF: Let $\mathcal{A}$ be a TRS, let $G = (V, E)$ be the graph of $\mathcal{A}$ ground terms, with V and E infinite. There must exist in $\mathcal{A}$ a signature symbol, say f, with arity $n \geq 1$ (otherwise the TRS would have a finite graph since the term set would be finite). Take $t_1, \ldots, t_{n-1}$ some $n-1$ fixed (possibly zero) $\mathcal{A}$ ground terms. Then consider the graph $G_f =_{def} (f(V), f(E))$ where $f(V) = \{f(t, t_1, \ldots, t_{n-1}) \mid t \in V\}$ and $f(E) = \{(f(d, t_1, \ldots, t_{n-1}), f(e, t_1, \ldots, t_{n-1})) \mid (d, e) \in E\}$. It is easy to see that G_f is isomorphic to G, and it is clearly not all of G nor empty. □

A canonical example is a complete infinite binary tree, where every branch is isomorphic to the entire tree. It is the graph of the (ground) TRS with unary symbols f and g, constant a and rules $a \to f(a)$ and $a \to g(a)$.

Note also that *the graph of ground terms* and *the graph of a ground TRS* are not the same thing. The property is sufficient but not necessary for the non-existence of a TRS whose graph is a given graph.

Thus Proposition 2.5.6 states that all interesting TRSs have this "fractal" property: the entire object is in some sense "equal to some of its parts". (In Appendix A we discuss "fractal objects" built from L-systems, which are in some way a variant of the STSs.)

Considering the entire graph of ground and non-ground terms, the same property holds almost trivially. According to the proof, if there were no symbol with arity ≥ 1, the only nodes are given by the set of all variables and constants. Since variables are denumerably infinite, this set is isomorphic to a proper subset. We will not consider this graph from now onwards.

But remark that this would not have worked if instead of considering sub-graphs one considers sub-ARSs. For instance, taking the TRS with unique rule $a \to f(a, a)$, let $(B, \to_B)$ be a proper sub-ARS. Then $a \notin B$ (otherwise $B = A$) therefore every term (node in the graph) has degree greater or equal than 2 (this is because the only node with degree 1 is the term a). Thus $(B, \to_B)$ cannot be isomorphic to $(A, \to_A)$.

The fact is that sub-ARSs are a less complex (more restrictive) substructure than sub-graphs, i.e. there are "much more" sub-graphs than sub-ARSs.

Corollary 2.5.7 *There are ARSs which are BD (hence FB) and which are not isomorphic to any TRS (either ground or not).*

PROOF: As an example take the following ARS $\mathcal{A} = (\mathbb{N} \cup \{*\}, \rightarrow)$ where $u \rightarrow v$ iff $u \in \mathbb{N}$ and, either $v = u + 1$, or $u = 1$ and $v = *$. Then it can be proved that the graph of $\mathcal{A}$ has no proper sub-graph isomorphic to itself, thus by the previous Proposition this ARS is not isomorphic to any TRS. □

2.5.3 Post canonical systems

Now we move the attention to *Post canonical systems* (PCS), as defined in the preliminaries chapter as well as in (14). From now onwards all PCSs will be finite, that is, with a finite set of rules and a finite alphabet. Like with TRSs, a PCS is left-linear (right-linear, linear) if its rules do not repeat variables (at the lhs, at the rhs, on neither side). In principle there is no restriction in having a lhs with only a single variable, contrarily to what TRSs require. Isomorphism between PCSs as well as between a PCS and another given rewriting system is defined as expected considering the reduction graphs. The FB and BD properties are also defined as usual for PCSs.

We have:

Lemma 2.5.8 *Every PCS is FB.*

PROOF: Every string may match a given PCS rule in a finite number of ways, and there is a finite number of rules to apply for each possible redex given by a substring. □

Lemma 2.5.9 *There are BD ARSs which are not isomorphic to any PCS.*

PROOF: Using a cardinality argument. There exist a non denumerable number of BD ARSs while the number of PCSs (with finite rules and symbols) is denumerably infinite. □

Lemma 2.5.10 *There are (linear) PCSs which are not isomorphic to any TRS (thus, in particular, there are (linear) PCSs which are not isomorphic to any STS).*

PROOF: Take the PCS with alphabet $\{a\}$ and rules

$a \rightarrow aa$

$a \rightarrow aaa$

$aaax \rightarrow aaaax$

It is clear that the reduction graph of the ground terms of this linear PCS (which is the same as the one in the proof of Corollary 2.5.7) is infinite and it is not isomorphic to any proper subgraph, then by Proposition 2.5.6 it is not the reduction graph of a TRS. □

Note that the idea of the proof of Lemma 2.5.10 does not work if one takes an alphabet of more than one character (the graph would be much more complex).

The literature also considers TRSs modulo associativity. One relation between PCSs and TRSs can be made explicit as follows.

Proposition 2.5.11 *Every PCS is isomorphic to some sub-ARS of some TRS modulo associativity.*

PROOF: (Sketch) Take a TRS with alphabet equal to the one of the PCS plus a special symbol \$ which will be used to delimit the strings during rewriting. And for each PCS rule

$$u_1 \dots u_m \to v_1 \dots v_m$$

consider the following TRS rule:

$$c(u_1, c(u_2, c(\dots u_m))) \to c(v_1, c(v_2, c(\dots v_m)))$$

where u_i and v_j might be constants or variables, and $c(\bullet, \bullet)$ is taken to be an associative binary function symbol. It is almost straightforward to verify that every reduction step in the PCS can be mimicked by a step in the sub-ARS given by the TRS restricted to the terms representing strings delimited by \$. □

Proposition 2.5.12 *Every TRS is isomorphic to a sub-ARS of some PCS.*

PROOF: (Sketch) Every term t of the TRS can be represented by a string s of the PCS whose alphabet is defined to include a symbol for each function symbol of the TRS (no matter which arity they have), to which the auxiliary symbols *(*, *)* and *,* (comma) are added. Note that the PCS will include ill-formed strings which do not correspond to any term of the TRS. Let L be the set of "legal" strings so formed. Take the PCS sub-ARS $\mathcal{S}(L)$. It is clearly isomorphic to the TRS. □

2.5.4 Summary

As a morale, since our goal was to justify the existence of the different rewriting formalisms, or to show that they are not isomorphic to each other, the previous statements prove the

Proposition 2.5.13 *The following inclusions of classes of rewriting systems hold and are proper:*

- *BD STS* $\subset$ *STS* $\subset$ *TRS / PCS* $\subset$ *FB ARS*
- *BD TRS* $\subset$ *TRS* $\subset$ *FB ARS*
- *BD ARS* $\subset$ *FB ARS* $\subset$ *ARS*

Note that most calculi with explicit substitution à la de Bruijn can be formulated as TRSs, therefore they are examples which rely in this hierarchy.

Although part of the above hierarchy comes from intuition, the question whether it is a simple, appropriate, pedagogical or research-oriented classification of rewriting paradigms will possibly remain open.

2.6 Conclusion

We have revisited the different rewriting paradigms: ARSs, TRSs, STSs and PCSs, looking for reasonable relations between them. The comparison of expressive power was made by using their reduction graphs. The relations involve sub-ARSs and quotients modulo equivalence. In the future we expect to continue this study, to treat other relations, and to look for other possible combinations of ARSs and characterizations of different rewriting systems. Further research includes to distinguish ground from non-ground TRSs, left-linearity and right-linearity, and the relationship with rational languages (84), with CSRSs and with graph-rewriting formulations (78).

We have found that bases, with its intuitive and natural meaning, do not exist in rewriting formalisms such as λ-calculus; not for the set of all terms, nor for the set of WN terms, nor for the set of SN terms, among other, which gives some evidence that these sets are indeed interesting.

We treat simulation questions in this thesis, although not with respect to any pair of formalisms. In a future approach we will consider a comparison not only based on the existence of isomorphisms between formalisms but on the existence of simulations between them, i.e. the possibility of translating derivations from one to another, no matter how many rewriting steps are done.

One goal of this chapter was to explore relations between the different rewriting formalisms, in order to find relations among them. Applications of these relations could be to decide whether a proof technique from one formalism may be carried to another one. Having said that, it is worth studying the relationship between different rewriting formalisms, in order to compare them and to be able to select one or other for expressing computing power.

ARSs are general enough in rewriting formulations, although they have little structure in general. Nevertheless they are the objects of a cartesian category, where the sub-ARSs are the sub-objects, and they form a complete lattice. As far as the author knows, no work has been done in studying sub-ARSs, ARSs products and ARSs of functions.

As a future task it remains to prove or disprove that there are (good) TRSs which are not isomorphic to any PCS. The proof of this fact seems to be more involved than the proofs we presented in this chapter.

We started and motivated the treatment of systems and subsystems here. In the subsequent chapters many important subsystems of several ARSs are analyzed, in particular interesting sets of terms of different λ-calculi having termination-related, confluence-related and typing-related properties, as we shall see. To illustrate most of the problems we work mainly with the $\lambda\upsilon$-calculus of explicit substitution since it has been accepted as a calculus with minimal rules and good properties.

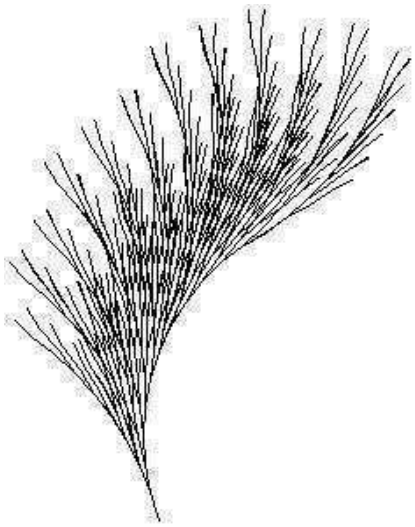


Figure 2.1: $S(SKK)(S(SKK))(S(SKK)(S(SKK)))$ after 70 left-most steps

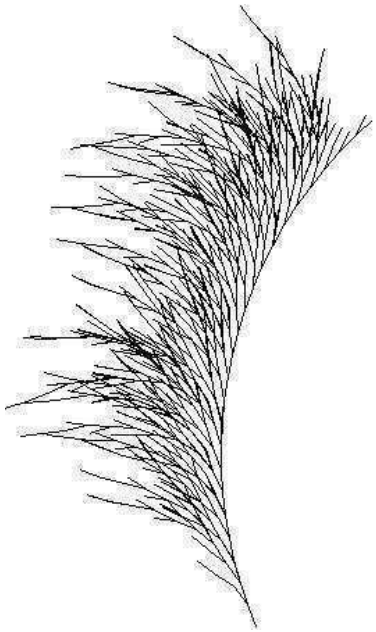


Figure 2.2: $(S(SSSS(SSSS)(SS)))S(SSS)$ after 65 left-most steps

Chapter 3

A λ-calculus without substitution: explicit substitution over pure terms

> *To the question whether we need the intuition for solving mathematical problems, it must be answered that in this case the language itself gives the necessary intuition ... The process of* calculus *actually provides that intuition.* – L. Wittgenstein, *Tractatus Logico-Philosophicus*

ABSTRACT In this chapter we study a λ-calculus with names and without substitutions, involving only four rules, which somehow result from embedding $\lambda\mathtt{x}$-calculus terms into classical λ-calculus terms. We show the relative soundness and confluence and discuss some of its advantages. We also propose two versions using de Bruijn indices, and prove that the main properties are preserved.

3.1 Introduction

This chapter revisits what is possibly the simplest λ-calculus and without substitutions at all (27; 66; 77) (and (73), which we could not access). The calculus has variable names and uses a minimal set of rules. We present some facts about such an a priori "simplistic" calculus over the set of classical λ-terms. We can say that it has no substitution at all, although another view indicates that it incorporates explicit substitution in some manner over the pure terms themselves. But this explicit substitution eliminates the need of formal (implicit, simultaneous) substitution. One of the main differences when comparing to other calculi of explicit substitution is that the β-rule is divided into four rules, and instead of rules propagating substitution, this "bookkeeping" is assigned in some way to the β-reduction itself in its present form, thus the non-necessity of a substitution operator. Moreover, without this substitution operator, some good properties of λ-calculus as well as other calculi of explicit substitution are preserved,

whereas others are not. In some sense this calculus can be seen as an interesting alternative to $\lambda\mathtt{x}$.

We also make some considerations on the implementation of this calculus. One of the features of such a calculus is that it needs not to be correct (sound) with respect to the β-reduction in the classical sense. That is, a derivation starting and ending in pure terms may not hold in λ-calculus, but this will not be an inconvenient if, given a term, the calculus includes a way to obtain its normal form when it exists.

We also contribute with a proposal of two versions of this calculus using de Bruijn indices, to which we transfer all the good properties of the initial calculus. Those versions can be seen as a starter's study in explicit substitution à la de Bruijn, since, as we will see throughout the chapter, it can be developed independently of the current literature about explicit substitution.

The plan of the chapter is as follows. Section 3.2 gives the syntax and rules of $\lambda_\emptyset$, the proposed calculus. In subsection 3.2.1 we prove the simulation of the β-reduction and the relative soundness of the rules. Subsection 3.2.2 addresses the confluence. In subsection 3.2.3 we discuss the relation with the λx calculus of explicit substitution. Subsection 3.2.4 treats typing considerations. In section 3.3 we present a de Bruijn version of $\lambda_\emptyset$. Subsection 3.4 shows the simulation and relative soundness for the de Bruijn setting. Subsection 3.4.1 treats its confluence. In section 3.5 we introduce another de Bruijn version and study its properties. Subsection 3.5.1 treats the simply typed version of this calculus. Soundness, simulation and preservation of strong normalization (PSN) are discussed in section 3.6. Finally, in section 3.7 we summarize our conclusion and identify future lines of work.

In what follows, when we mention $\lambda\mathtt{x}$ actually we will mean $\lambda\mathtt{x}^{-}$.

3.2 The $\lambda_\emptyset$-calculus

We begin by introducing the $\lambda_\emptyset$-calculus. Its syntax and rules are given next.

Definition 3.2.1 *The $\lambda_\emptyset$-calculus is a λ-calculus with names, over the same syntax of classical λ-calculus. Its four rules are:*

$$\begin{array}{llll}
(\lambda x.x)M & \to_{\lambda var1} & M & \\
(\lambda x.y)M & \to_{\lambda var2} & y & x \neq y \\
(\lambda x.PQ)M & \to_{\lambda app} & (\lambda x.P)M((\lambda x.Q)M) & \\
(\lambda x.(\lambda y.P))M & \to_{\lambda\lambda} & \lambda y.(\lambda x.P)M & x \neq y
\end{array}$$

Let us denote with NF_β the set of β-normal forms, with $NF_{\lambda_\emptyset}$ the set of $\lambda_\emptyset$-normal forms, with WN_β the set of β-weakly normalizing terms, with $WN_{\lambda_\emptyset}$ the set of $\lambda_\emptyset$-weakly normalizing terms, with SN_β the set of β-strongly normalizing terms and with $SN_{\lambda_\emptyset}$ the set of $\lambda_\emptyset$-strongly normalizing terms.

In the last rule we assume the usual free variable convention.

The reader may note an analogy with the rules of the substitution calculus associated to the $\lambda\mathtt{x}$-calculus of explicit substitution (see the preliminaries). Having said this, we will use $\lambda\mathtt{x}$ as a reference for comparison, but we do not need to transfer its properties to our calculus. The benefit of achieving this is that we will have a calculus over the pure (classical) terms without the need of a closure operator and with less rules than $\lambda\mathtt{x}$.

Apart from being inspired in $\lambda\mathtt{x}$, the first 3 rules resemble respectively the effects of the combinatory logic rules for I, K and S (11). The fourth rule has no counterpart in combinatory logic.

As with classical λ-calculus we have

Remark 3.2.2 *If $M \to_{\lambda_\emptyset} N$ then $FV(N) \subseteq FV(M)$.*

Proof: By induction on M. □

Another feature is that in $\lambda_\emptyset$ there is no clear distinction between the β-reduction and the associated substitution calculus. With this formulation of a calculus we intend to show how with just classical λ-terms the same work of an explicit substitution calculus is performed (since it has collapsed with the β-rule). In some sense, this is a calculus without substitutions (and of course this means not to use any closure operator).

Our proofs for $\lambda_\emptyset$ do not use $\lambda\mathtt{x}$ properties. We prove our results independently, using only classical results from λ-calculus.

3.2.1 Simulation and soundness

We now show that our calculus enjoys simulation of the β-reduction.

Proposition 3.2.3 (Simulation) *If $M \to_\beta N$ then $M \overset{+}{\to}_{\lambda_\emptyset} N$. Furthermore, if $M \overset{*}{\to}_\beta N$ then $M \overset{*}{\to}_{\lambda_\emptyset} N$.*

Proof: By induction on the position of the redex.

- if the reduction occurs at the root, we have that $M = (\lambda x.U)V \to_\beta U\{x \leftarrow V\}$, then the following cases may occur:
  - $U = x$, then $M \to_{\lambda var1} V = U\{x \leftarrow V\}$
  - $U = y$, then $M \to_{\lambda var2} y = U\{x \leftarrow V\}$
  - $U = PQ$, then $M \to_{\lambda app} (\lambda x.P)V((\lambda x.Q)V) \overset{+}{\to}^{IH}_{\lambda_\emptyset} P\{x \leftarrow V\}Q\{x \leftarrow V\}$ $= U\{x \leftarrow V\}$
  - $U = \lambda y.P$, then $M \to_{\lambda\lambda} \lambda y.(\lambda x.P)V \overset{+}{\to}^{IH}_{\lambda_\emptyset} \lambda y.P\{x \leftarrow V\} = U\{x \leftarrow V\}$

- if the reduction is internal, then the following cases may occur:
  - $PQ \to_\beta P'Q$ with $P \to_\beta P'$, then by IH $P \xrightarrow{+}_{\lambda_\emptyset} P'$ thus $PQ \xrightarrow{+}_{\lambda_\emptyset} P'Q$
  - $PQ \to_\beta PQ'$ with $Q \to_\beta Q'$, analogous to the previous case
  - $\lambda x.P \to_\beta \lambda x.P'$ with $P \to_\beta P'$, then by IH $P \xrightarrow{+}_{\lambda_\emptyset} P'$ thus $\lambda x.P \xrightarrow{+}_{\lambda_\emptyset} \lambda x.P'$

The second statement follows immediately by induction on the length of the derivation. □

It is interesting that this calculus is not "sound", since it does not satisfy that when $M \to_{\lambda_\emptyset} N$ then $M \xrightarrow{*}_\beta N$. This happens because the right-hand side of the λapp-rule (respectively, the $\lambda\lambda$-rule) in general is not a β-reduct of the left-hand side. But it does satisfy *relative soundness* (Lemma 3.2.4 and Corollary 3.2.8).

Lemma 3.2.4 (Soundness with respect to $=_\beta$) *If $M \xrightarrow{*}_{\lambda_\emptyset} N$ then $M =_\beta N$*

PROOF: We prove that if $M \to_{\lambda_\emptyset} N$ then $M =_\beta N$, by induction on the position of the redex. Then the result will follow by induction on the length of the $\xrightarrow{*}_{\lambda_\emptyset}$-derivation.

- if the reduction occurs at the root, then the following cases may occur:
  - the reduction is $(\lambda x.x)M \to_{\lambda var1} M$, then clearly $(\lambda x.x)M \to_\beta M$
  - the reduction is $(\lambda x.y)M \to_{\lambda var2} y$, then clearly $(\lambda x.y)M \to_\beta y$
  - the reduction is $(\lambda x.PQ)M \to_{\lambda app} (\lambda x.P)M((\lambda x.Q)M)$, then $(\lambda x.PQ)M \to_\beta P\{x \leftarrow M\}Q\{x \leftarrow M\}$ and $(\lambda x.P)M((\lambda x.Q)M) \xrightarrow{*}_\beta P\{x \leftarrow M\}Q\{x \leftarrow M\}$ so we are done
  - the reduction is $(\lambda x.(\lambda y.P))M \to_{\lambda\lambda} \lambda y.(\lambda x.P)M$, then $(\lambda x.(\lambda y.P))M \to_\beta \lambda y.P\{x \leftarrow M\}$ and $\lambda y.(\lambda x.P)M \to_\beta \lambda y.P\{x \leftarrow M\}$ so again we are done.
- if the reduction is internal, then the following cases may occur:
  - $PQ \to_{\lambda_\emptyset} P'Q$ with $P \to_{\lambda_\emptyset} P'$, then by IH $P =_\beta P'$ thus $PQ =_\beta P'Q$
  - $PQ \to_{\lambda_\emptyset} PQ'$ with $Q \to_{\lambda_\emptyset} Q'$, analogous to the previous case
  - $\lambda x.P \to_{\lambda_\emptyset} \lambda x.P'$ with $P \to_{\lambda_\emptyset} P'$, then by IH $P =_\beta P'$ thus $\lambda x.P =_\beta \lambda x.P'$

□

3.2.2 Confluence

A natural question is: given that the β-reduction is split into four rules, does one lose the confluence of the classical λ-calculus? The answer is given by the following Corollary. Note that confluence of $\lambda_\emptyset$ is not an immediate consequence of being orthogonal, i.e. left-linear and

without critical pairs, in its higher-order formulation (14; 81). The calculus is clearly left-linear but, unlike λ-calculus and combinatory logic, it has critical pairs, thus we cannot use the higher-order result that orthogonality implies confluence. The same observation applies for the calculi introduced later in this chapter.

Corollary 3.2.5 (Confluence of $\lambda_\emptyset$) *If $M \xrightarrow{*}_{\lambda_\emptyset} M_1$ and $M \xrightarrow{*}_{\lambda_\emptyset} M_2$ then there exists M_3 such that $M_1 \xrightarrow{*}_{\lambda_\emptyset} M_3$ and $M_2 \xrightarrow{*}_{\lambda_\emptyset} M_3$.*

PROOF: Let $M \xrightarrow{*}_{\lambda_\emptyset} M_1$ and $M \xrightarrow{*}_{\lambda_\emptyset} M_2$, then by Lemma 3.2.4 $M =_\beta M_1$ and $M =_\beta M_2$, then $M_1 =_\beta M_2$, which by the confluence of λ-calculus implies that there exists M_3 such that $M_1 \xrightarrow{*}_\beta M_3$ and $M_2 \xrightarrow{*}_\beta M_3$. By Proposition 3.2.3, $M_1 \xrightarrow{*}_{\lambda_\emptyset} M_3$ and $M_2 \xrightarrow{*}_{\lambda_\emptyset} M_3$. □

Corollary 3.2.6 *The relations $=_\beta$ and $=_{\lambda_\emptyset}$ coincide.*

PROOF: Consequence of Proposition 3.2.3 and Lemma 3.2.4. □

We show that the set of $\lambda_\emptyset$-normal forms coincide with the set of normal forms of the classical λ-calculus.

Lemma 3.2.7 ($\lambda_\emptyset$-normal forms) $NF_\beta = NF_{\lambda_\emptyset}$

PROOF: If $M \notin NF_{\lambda_\emptyset}$ then there is a $\lambda_\emptyset$-redex, which is clearly also a β-redex, thus $M \notin NF_\beta$. If $M \notin NF_\beta$ then there is a subterm $(\lambda x.U)V \subseteq M$ which is a β-redex. Reasoning over U, whether it is a variable, an application or an abstraction, in either case we have that $(\lambda x.U)V$ is also a $\lambda_\emptyset$-redex, thus $M \notin NF_{\lambda_\emptyset}$. □

As a consequence we have

Corollary 3.2.8 (Soundness with respect to normal forms) *If $M \xrightarrow{*}_{\lambda_\emptyset} N$ and $N \in NF_{\lambda_\emptyset}$ then $M \xrightarrow{*}_\beta N$*

PROOF: If $M \xrightarrow{*}_{\lambda_\emptyset} N \in NF_{\lambda_\emptyset}$, by Lemma 3.2.4 $M =_\beta N$. By Lemma 3.2.7, $N \in NF_\beta$ thus $M \xrightarrow{*}_\beta N$. □

The above Corollary states that we expect sound calculations when reaching normal forms. As another nice consequence we have

Corollary 3.2.9 *1.* $WN_\beta = WN_{\lambda_\emptyset}$

2. $SN_\beta \subset WN_{\lambda_\emptyset}$

PROOF:

1. If $M \xrightarrow{*}_{\beta} N \in NF_{\beta}$ then $M \in WN_{\lambda_\emptyset}$ since $NF_{\beta} = NF_{\lambda_\emptyset}$ and $M \xrightarrow{*}_{\lambda_\emptyset} N$ (Proposition 3.2.3). If $M \xrightarrow{*}_{\lambda_\emptyset} N \in NF_{\lambda_\emptyset}$ then by Corollary 3.2.8 $M \xrightarrow{*}_{\beta} N$ and again use that $NF_{\beta} = NF_{\lambda_\emptyset}$.
2. Because $SN_{\beta} \subset WN_{\beta}$ and using item (1).

□

3.2.3 Relation with $\lambda\mathtt{x}$

In this subsection we exhibit a mapping which will show that $\lambda_\emptyset$ behaves "almost" as $\lambda\mathtt{x}$.

Definition 3.2.10 *We define the mapping* $[\bullet] : \Lambda x \to \Lambda$ *as follows:*
$[x] = x$
$[MN] = [M][N]$
$[\lambda x.M] = \lambda x.[M]$
$[M < x = N >] = (\lambda x.[M])[N]$

Remark 3.2.11 *1.* $[\bullet]$ *is surjective and non-injective*

2. for every $M \in \Lambda x$, $[M] = M$ *iff* $M \in \Lambda$ *(i.e. only pure* $\lambda\mathtt{x}$*-terms map to themselves)*

3. in particular, for every $M \in \Lambda x$, $[[M]] = [M]$

Proof: Straightforward. □

The following Proposition states that $[\bullet]$ is an *homomorphism* from $\lambda\mathtt{x}$-calculus to $\lambda_\emptyset$-calculus, which restricted to the set Λ is the identity[1].

Actually we already have an homomorphism from $\lambda\mathtt{x}$ to $\lambda_\emptyset$, namely $x(\bullet)$, i.e. to take the x-normal form. Because $M \xrightarrow{*}_{\lambda\mathtt{x}} N$ implies (see (15)) $x(M) \xrightarrow{*}_{\beta} x(N)$ and then by simulation of the β-reduction we have $x(M) \xrightarrow{*}_{\lambda_\emptyset} x(N)$, thus $x(\bullet)$ is an homomorphism which also leaves unchanged the pure terms. We simply show that $[\bullet]$ is another such homomorphism.

Proposition 3.2.12 (Simulation of $\lambda\mathtt{x}$) *Let* $M, N \in \Lambda x$.

1. If $M \to_{\lambda\mathtt{x}} N$ *then* $[M] \xrightarrow{*}_{\lambda_\emptyset} [N]$

2. If $M \xrightarrow{*}_{\lambda\mathtt{x}} N$ *then* $[M] \xrightarrow{*}_{\lambda_\emptyset} [N]$

Proof:

[1] Without this extra requirement, or an equivalent one, trivial homomorphisms always exist such as a constant function.

1. By induction on the position of the redex.

   - if the reduction occurs at the root, then the following cases may occur:
     - the reduction is $(\lambda x.M)N \rightarrow_{Beta} M < x = N >$, then $[(\lambda x.M)N] = (\lambda x.[M])[N]$ $= [M < x = N >]$
     - the reduction is $(PQ) < x = N > \rightarrow_{App} P < x = N > Q < x = N >$, then $[(PQ) < x = N >] = (\lambda x.[P][Q])[N] \rightarrow_{\lambda app} (\lambda x.[P])[N]((\lambda x.[Q])[N])$ $= [(\lambda x.P)N((\lambda x.Q)N)]$
     - the reduction is $(\lambda y.M) < x = N > \rightarrow_{Lam} \lambda y.M < x = N >$, then $[(\lambda y.M) < x = N >] = (\lambda x.(\lambda y.[M]))[N]) \rightarrow_{\lambda\lambda} \lambda y.(\lambda x.[M])[N] = [\lambda y.(\lambda x.M)N]$
     - the reduction is $x < x = N > \rightarrow_{var1} N$, then $[x < x = N >] = (\lambda x.x)[N]$ $\rightarrow_{\lambda var1} [N]$
     - the reduction is $y < x = N > \rightarrow_{var2} y$, then $[y < x = N >] = (\lambda x.y)[N]$ $\rightarrow_{\lambda var2} y = [y]$
   - if the reduction is internal, then the following cases may occur:
     - $PQ \rightarrow_{\lambda\mathtt{x}} P'Q$ with $P \rightarrow_{\lambda\mathtt{x}} P'$, then by IH $[P] \stackrel{*}{\rightarrow}_{\lambda_\emptyset} [P']$ thus $[PQ] = [P][Q] \stackrel{*}{\rightarrow}_{\lambda_\emptyset} [P'][Q] = [P'Q]$
     - $PQ \rightarrow_{\lambda\mathtt{x}} PQ'$ with $Q \rightarrow_{\lambda\mathtt{x}} Q'$, analogous to the previous case
     - $\lambda x.P \rightarrow_{\lambda\mathtt{x}} \lambda x.P'$ with $P \rightarrow_{\lambda\mathtt{x}} P'$, then by IH $[P] \stackrel{*}{\rightarrow}_{\lambda_\emptyset} [P']$ thus $[\lambda x.P] = \lambda x.[P] \stackrel{*}{\rightarrow}_{\lambda_\emptyset} \lambda x.[P'] = [\lambda x.P']$
     - $P < x = N > \rightarrow_{\lambda\mathtt{x}} P' < x = N >$ with $P \rightarrow_{\lambda\mathtt{x}} P'$, then by IH $[P] \stackrel{*}{\rightarrow}_{\lambda_\emptyset} [P']$ thus $[P < x = N >] = (\lambda x.[P])[N] \stackrel{*}{\rightarrow}_{\lambda_\emptyset} (\lambda x.[P'])[N] = [P' < x = N >]$
     - $P < x = N > \rightarrow_{\lambda\mathtt{x}} P < x = N' >$ with $N \rightarrow_{\lambda\mathtt{x}} N'$, then by IH $[N] \stackrel{*}{\rightarrow}_{\lambda_\emptyset} [N']$ thus $[P < x = N >] = (\lambda x.[P])[N] \stackrel{*}{\rightarrow}_{\lambda_\emptyset} (\lambda x.[P])[N'] = [P < x = N' >]$

2. Using the previous item and induction on the length of the derivation.

□

In this way we can embed $\lambda\mathtt{x}$ onto $\lambda_\emptyset$. It is worth mention that we did not find, whether it exists, a variant of this calculus which satisfies the stronger property: $M \rightarrow_{\lambda\mathtt{x}} N$ then $[M] \stackrel{+}{\rightarrow}_{\lambda_\emptyset} [N]$.

3.2.4 Typing and PSN

Without details we remark that $\lambda_\emptyset$ admits simple typing in the same way as classical λ-calculus. But we will see with a simple example that $\lambda_\emptyset$ does not satisfy PSN. That is, there are terms $M \in \Lambda$ which are SN in λ-calculus but may admit an infinite derivation in $\lambda_\emptyset$ by means of these four rules, actually the application of rules λapp and $\lambda\lambda$ may alternate infinitely

often. The example of infinite derivation, as pointed out in (66), begins with a term of the form $(\lambda x.(\lambda y.P)Q)R$, and roughly looks like follows:

$$\begin{array}{lll}
(\lambda x.(\lambda y.P)Q)R & \to_{\lambda app} & ((\lambda x.(\lambda y.P))R)(\dots) \\
& \to_{\lambda\lambda} & (\lambda y.(\lambda x.P)R)(\dots) \\
& \to_{\lambda app} & ((\lambda y.(\lambda x.P))(\dots))((\dots)(\dots)) \\
& \to_{\lambda\lambda} & (\lambda x.(\lambda y.P)(\dots))(\dots) \\
& \to_{\lambda app} & ((\lambda x.(\lambda y.P))(\dots))((\dots)(\dots)) \\
& \to_{\lambda\lambda} & (\lambda y.(\lambda x.P)(\dots))(\dots) \\
& \dots &
\end{array}$$

where the application of both rules alternate and occur at the left-most positions. Therefore, even when we cannot partition the $\lambda_\emptyset$-rules set into a (Beta)-rule on one side and a substitution calculus on the other, it is immediate that PSN does not hold, since in this example the meta-variables P, Q and R can be instantiated in such a way that the above term is SN in classical λ-calculus, for example $P = Q = R = \lambda x.x$. As a side note, this also shows that the left-most reduction strategy is not standard in $\lambda_\emptyset$, i.e. does not yield a normal form when it exists, contrarily to the case of the classical λ-calculus.

It might be thought that it is the price of not having in the syntax a distinction between closure and abstraction, thus the explicit substitution idea becomes useful, and it can be seen as some way to avoid this phenomenon (although this has not been the reason for its introduction from a historical point of view; see the conclusion).

Since PSN does not hold, it is not plausible to prove strong normalization of simply typed terms by transferring the problem to classical λ-calculus. The most that one can guarantee is that a typed term is in $WN_\beta (= WN_{\lambda_\emptyset})$. This holds because of the inclusion $SN_\beta \subset WN_{\lambda_\emptyset}$ (Corollary 3.2.9) and the fact that simply typed terms in λ-calculus are SN. We will see a more detailed treatment of typing in subsection 3.5.1.

3.3 A de Bruijn calculus based on $\lambda_\emptyset$

In this section we study a de Bruijn version of $\lambda_\emptyset$, over the set of pure de Bruijn terms. This calculus is motivated by the fact that the problem of α-conversion from classical λ-calculus remains. This is not the only possibility of such a calculus as we shall see. We introduce here the *exchange operators* e_k, to be explained below.

Definition 3.3.1 *Our de Bruijn formulation will have the same syntax of the (pure) de Bruijn terms, and the following rules*

$$\begin{array}{lll}
(\lambda 1)a & \to_{db\lambda var1} & a \\
(\lambda(m+1))a & \to_{db\lambda var2} & m \\
(\lambda ab)c & \to_{db\lambda app} & (\lambda a)c((\lambda b)c) \\
(\lambda(\lambda a))c & \to_{db\lambda\lambda} & \lambda(\lambda e_1(a))\varphi_0(c)
\end{array}$$

where $\varphi_0(\bullet) = U_0^2(\bullet)$ *is the usual (implicit, simultaneous) updating operator (see the preliminaries), and* $e_k(\bullet)$ *is the* $(k, k+1)$*-exchange operator, defined by the equalities*

$$\begin{array}{rcl} e_k(m) & = & \begin{cases} k+1 & \text{if } m = k \\ k & \text{if } m = k+1 \\ m & \text{if } m \neq k, k+1 \end{cases} \\ e_k(\lambda a) & = & \lambda e_{k+1}(a) \\ e_k(ab) & = & e_k(a)e_k(b) \end{array}$$

We will call $\lambda_{\emptyset dB}$ *the preceding calculus over de Bruijn terms.*

Thus $\lambda_{\emptyset dB}$ seems to be the simplest of all de Bruijn calculi of "explicit substitution" (although the operators φ_k and e_k are not included in its syntax). In the following sections we address the main properties of this calculus, and we will see that the essential properties of $\lambda_\emptyset$ are preserved.

Let us denote with $NF_{\lambda_{\emptyset dB}}$ the set of $\lambda_{\emptyset dB}$-normal forms, with $WN_{\lambda_{\emptyset dB}}$ the set of $\lambda_{\emptyset dB}$-weakly normalizing terms and with $SN_{\lambda_{\emptyset dB}}$ the set of $\lambda_{\emptyset dB}$-strongly normalizing terms.

3.4 Simulation and soundness of $\lambda_{\emptyset dB}$

Since mappings between $\lambda_\emptyset$ and $\lambda_{\emptyset dB}$ do not seem at hand, we plan to show the simulation and relative soundness using the classical isomorphism as it is done for classical λ-calculus with respect to the de Bruijn formulation. We remark that the proofs may not use procedures similar to those on section 3.2.1.

Therefore we will now translate the embedding between calculi with names to an embedding between de Bruijn calculi. We recall the isomorphism between classical λ-calculus and the λ-calculus à la de Bruijn, given by the functions $w_{[x_1,\ldots,x_n]}(\bullet) : \Lambda \to \Lambda_{dB}$ and $u_{[x_1,\ldots,x_n]}(\bullet) : \Lambda_{dB} \to \Lambda$ (see the preliminaries). If the context makes it clear, we will not write as a subscript the ordered set of free variables $[x_1, \ldots, x_n]$ of the term to which w applies to. We will only use the equality $w(u(a)) = a$ and not the other equivalence in the following treatment.

Lemma 3.4.1 *For every* $M \in \Lambda$*, we have that*
$e_k(w_{[x_1,\ldots,x_{k-1},x,y,x_{k+2},\ldots,x_n]}(M)) = w_{[x_1,\ldots,x_{k-1},y,x,x_{k+2},\ldots,x_n]}(M)$.

Proof: By induction on M.

- $M = x$, then $e_k(w_{[x_1,\ldots,x_{k-1},x,y,x_{k+2},\ldots,x_n]}(M)) = e_k(k) = k+1 = w_{[x_1,\ldots,x_{k-1},y,x,x_{k+2},\ldots,x_n]}(M)$ $(x \neq y)$.
- $M = y$, then $e_k(w_{[x_1,\ldots,x_{k-1},x,y,x_{k+2},\ldots,x_n]}(M)) = e_k(k+1) = k = w_{[x_1,\ldots,x_{k-1},y,x,x_{k+2},\ldots,x_n]}(M)$.
- $M = x_m \neq x, y$, then $e_k(w_{[x_1,\ldots,x_{k-1},x,y,x_{k+2},\ldots,x_n]}(M)) = e_k(m) = m = w_{[x_1,\ldots,x_{k-1},y,x,x_{k+2},\ldots,x_n]}(M)$.

- $M = \lambda z.P$, then $e_k(w_{[x_1,\ldots,x_{k-1},x,y,x_{k+2},\ldots,x_n]}(M)) = e_k(\lambda w_{[z,x_1,\ldots,x_{k-1},x,y,x_{k+2},\ldots,x_n]}(P)) = \lambda e_{k+1}(w_{[z,x_1,\ldots,x_{k-1},x,y,x_{k+2},\ldots,x_n]}(P)) =_{IH} \lambda w_{[z,x_1,\ldots,x_{k-1},y,x,x_{k+2},\ldots,x_n]}(P) = w_{[x_1,\ldots,x_{k-1},y,x,x_{k+2},\ldots,x_n]}(M)$.
- $M = PQ$, then $e_k(w_{[x_1,\ldots,x_{k-1},x,y,x_{k+2},\ldots,x_n]}(M)) = e_k(w_{[x_1,\ldots,x_{k-1},x,y,x_{k+2},\ldots,x_n]}(P))\ e_k(w_{[x_1,\ldots,x_{k-1},x,y,x_{k+2},\ldots,x_n]}(Q)) =_{IH} w_{[x_1,\ldots,x_{k-1},y,x,x_{k+2},\ldots,x_n]}(P)\ w_{[x_1,\ldots,x_{k-1},y,x,x_{k+2},\ldots,x_n]}(Q) = w_{[x_1,\ldots,x_{k-1},y,x,x_{k+2},\ldots,x_n]}(M)$.

□

Lemma 3.4.2 *For every $M \in \Lambda$, $\varphi_0(w_{[x_1,\ldots,x_n]}(M)) = w_{[y,x_1,\ldots,x_n]}(M)$ if $y \notin FV(M)$.*

PROOF: See (43) for a proof of the more general property $U^i_k(w_{[x_1,\ldots,x_k,x_{k+i},\ldots,x_n]}(M)) = w_{[x_1,\ldots,x_n]}(M)$ for every term M provided $k \geq 0$, $i \geq 1$, $n \geq k+i$ and $x_{k+1},\ldots,x_{k+i-1} \notin FV(M)$. Here the interesting case is $\varphi_0(w_{[x_1,\ldots,x_n]}(x_i)) = \varphi_0(i) = i+1 = w_{[y,x_1,\ldots,x_n]}(x_i)$. □

Lemma 3.4.3 *Let $M, N \in \Lambda$.*

1. *if $M \to_{\lambda_\emptyset} N$ then $w(M) \to_{\lambda_{\emptyset dB}} w(N)$*
2. *if $M \overset{+}{\to}_{\lambda_\emptyset} N$ then $w(M) \overset{+}{\to}_{\lambda_{\emptyset dB}} w(N)$*

PROOF:

1. By induction on the position of the redex.
   - if the reduction occurs at the root, then the following cases may occur:
     - the reduction is $(\lambda x.x)N \to_{\lambda var1} N$, then $w((\lambda x.x)N) = (\lambda 1)w(N) \to_{db\lambda var1} w(N)$
     - the reduction is $(\lambda x.y)N \to_{\lambda var2} y$, then $w((\lambda x.y)N) = (\lambda(m+1))w(N) \to_{db\lambda var1} m = w(y)$
     - the reduction is $(\lambda x.PQ)N \to_{\lambda app} (\lambda x.P)N((\lambda x.Q)N)$, then $w_{[x_1,\ldots,x_n]}((\lambda x.PQ)N) = (\lambda w_{[x,x_1,\ldots,x_n]}(P)w_{[x,x_1,\ldots,x_n]}(Q))\ w_{[x_1,\ldots,x_n]}(N) \to_{db\lambda app} (\lambda w_{[x,x_1,\ldots,x_n]}(P))w_{[x_1,\ldots,x_n]}(N)((\lambda w_{[x,x_1,\ldots,x_n]}(Q))w_{[x_1,\ldots,x_n]}(N)) = w_{[x_1,\ldots,x_n]}((\lambda x.P)N((\lambda x.Q)N))$.
     - the reduction is $(\lambda x.(\lambda y.P))N \to_{\lambda\lambda} \lambda y.(\lambda x.P)N$, then $w_{[x_1,\ldots,x_n]}((\lambda x.(\lambda y.P))N) = (\lambda(\lambda w_{[y,x,x_1,\ldots,x_n]}(P)))w_{[x_1,\ldots,x_n]}(N) \to_{db\lambda\lambda} \lambda(\lambda e_1(w_{[y,x,x_1,\ldots,x_n]}(P))\varphi_0(w_{[x_1,\ldots,x_n]}(N)) = \lambda(\lambda w_{[x,y,x_1,\ldots,x_n]}(P))w_{[y,x_1,\ldots,x_n]}(N))$ (by Lemmas 3.4.1 and 3.4.2) $= w_{[x_1,\ldots,x_n]}(\lambda y.(\lambda x.P)N)$.

- if the reduction is internal, the proof proceeds in a straightforward way.

2. By item (1) and induction on the length of the derivation.

□

As a consequence we have that $\lambda_{\emptyset dB}$ also verifies simulation of the β_{dB}-reduction.

Proposition 3.4.4 *Let a, b be de Bruijn terms. If $a \to_{\beta_{dB}} b$ then $a \xrightarrow{+}_{\lambda_{\emptyset dB}} b$*

PROOF: Suppose $a \to_{\beta_{dB}} b$. Then by the isomorphism, $u(a) \to_{\beta} u(b)$. By simulation of the β-reduction, $u(a) \xrightarrow{+}_{\lambda_{\emptyset}} u(b)$. By Lemma 3.4.3 (2), $w(u(a)) \xrightarrow{+}_{\lambda_{\emptyset dB}} w(u(b))$. Since w and u are inverses of each other, the result follows. □

Again relative soundness holds.

Lemma 3.4.5 (Soundness with respect to $=_{\beta_{dB}}$) *If $a \xrightarrow{*}_{\lambda_{\emptyset dB}} b$ then $a =_{\beta_{dB}} b$*

PROOF: Similar to Lemma 3.2.4. □

3.4.1 Confluence of $\lambda_{\emptyset dB}$

As before, we have the confluence of $\lambda_{\emptyset dB}$ (on closed terms) and the subsequent results.

Corollary 3.4.6 (Confluence of $\lambda_{\emptyset dB}$) *If $a \xrightarrow{*}_{\lambda_{\emptyset dB}} a_1$ and $a \xrightarrow{*}_{\lambda_{\emptyset dB}} a_2$ then there exists a_3 such that $a_1 \xrightarrow{*}_{\lambda_{\emptyset dB}} a_3$ and $a_2 \xrightarrow{*}_{\lambda_{\emptyset dB}} a_3$.*

PROOF: Similar to Lemma 3.2.5. □

Corollary 3.4.7 *The relations $=_{\beta dB}$ and $=_{\lambda_{\emptyset dB}}$ coincide.*

PROOF: Similar to Corollary 3.2.6 □

Lemma 3.4.8 ($\lambda_{\emptyset dB}$-normal forms) $NF_{\beta_{dB}} = NF_{\lambda_{\emptyset dB}}$

PROOF: Similar to Lemma 3.2.7. □

Corollary 3.4.9 (Soundness with respect to normal forms) *If $M \xrightarrow{*}_{\lambda_{\emptyset dB}} N$ and $N \in NF_{\lambda_{\emptyset dB}}$ then $M \xrightarrow{*}_{\beta_{dB}} N$*

PROOF: Similar to Corollary 3.2.8. □

Corollary 3.4.10 *1.* $WN_{\beta_{dB}} = WN_{\lambda_{\emptyset dB}}$

2. $SN_{\beta_{dB}} \subset WN_{\lambda_{\emptyset dB}}$

PROOF: Similar to Corollary 3.2.9. □

3.5 φ and e as explicit operators

Even when the initial motivation of $\lambda_\emptyset$ was to get rid of substitution, we can "go back" and also formulate a de Bruijn version of $\lambda_\emptyset$ by being explicit in the definition of U_k^2 and e_k, that is we can include them in the term syntax and the calculus itself may have rules for one or -better- both of them. The resulting calculus is the following.

Definition 3.5.1 *The $\lambda_{\emptyset S}$-calculus has the following syntax:*

$$a \quad ::= \quad n \mid (aa) \mid (\lambda a) \mid \varphi'_k(a) \mid e'_j(a) \quad k \geq 0,\ j \geq 1,\ n \geq 1$$

and the following rules:

$$\begin{array}{lll}
(\lambda 1)a & \to_{s\lambda var1} & a \\
(\lambda(m+1))a & \to_{s\lambda var2} & m \\
(\lambda ab)c & \to_{s\lambda app} & (\lambda a)c((\lambda b)c) \\
(\lambda(\lambda a))c & \to_{s\lambda\lambda} & \lambda(\lambda e'_1(a))\varphi'_0(c) \\
\varphi'_k(m) & \to_{\varphi var} & \begin{cases} m+1 & \text{if } m > k \\ m & \text{if } m \leq k \end{cases} \\
\varphi'_k(\lambda a) & \to_{\varphi\lambda} & \lambda\varphi'_{k+1}(a) \\
\varphi'_k(ab) & \to_{\varphi app} & \varphi'_k(a)\varphi'_k(b) \\
e'_k(m) & \to_{evar} & \begin{cases} k+1 & \text{if } m = k \\ k & \text{if } m = k+1 \\ m & \text{if } m \neq k, k+1 \end{cases} \\
e'_k(\lambda a) & \to_{e\lambda} & \lambda e'_{k+1}(a) \\
e'_k(ab) & \to_{eapp} & e'_k(a)e'_k(b)
\end{array}$$

The set of terms will be denoted by Λ_{dBs}. The set of $\lambda_{\emptyset S}$-normal forms will be denoted by $NF_{\lambda_{\emptyset S}}$

Remark that we use φ'_k and e'_j (with the *primes*) in order to distinguish them from last section's φ_k and e_j implicit operators, which we will be still using.

Definition 3.5.2 *We denote with $\emptyset_S$ the calculus formed by rules $(\varphi var), (\varphi\lambda), (\varphi app), (evar), (e\lambda)$, $(eapp)$, i.e. rules governing the propagation of the operators φ'_k and e'_j.*

In the rest of this subsection we will study the properties of $\lambda_{\emptyset S}$.

Lemma 3.5.3 *The $\emptyset_S$-calculus is SN*

PROOF: Standard, using the recursive path order method (9; 14) taking $e'_j, \varphi'_k >> \lambda, @$. □

Lemma 3.5.4 *The $\emptyset_S$-calculus is WCR*

PROOF: There are no critical pairs between rules $(\varphi var), (\varphi\lambda), (\varphi app), (evar), (e\lambda), (eapp)$. □

Remark 3.5.5 *The set of $\emptyset_S$-normal forms is equal to Λ_{dB}.*

PROOF: Every term with an occurrence of the φ'_k or e'_j operators will match some of the $\emptyset_S$-rules. □

Corollary 3.5.6 *The $\emptyset_S$-calculus is CR*

PROOF: By Lemmas 3.5.3 and 3.5.4, and Newman's Lemma. □

We denote with $\emptyset_S(a)$ the unique $\emptyset_S$-normal form of a term $a \in \Lambda_{\emptyset S}$.
The rest of the subsection will prove the confluence of $\lambda_{\emptyset S}$ i.e. the full calculus.

Lemma 3.5.7 *1.* $\emptyset_S(\lambda a) = \lambda\emptyset_S(a)$

2. $\emptyset_S(ab) = \emptyset_S(a)\emptyset_S(b)$

PROOF:

1. $\lambda a \xrightarrow{*}_{\emptyset_S} \lambda\emptyset_S(a)$ which is clearly an $\emptyset_S$-normal form. By CR, $\emptyset_S(\lambda a) = \lambda\emptyset_S(a)$.
2. Analogous to item (1): $ab \xrightarrow{*}_{\emptyset_S} \emptyset_S(a)\emptyset_S(b)$ which is clearly an $\emptyset_S$-normal form. By CR, $\emptyset_S(ab) = \emptyset_S(a)\emptyset_S(b)$.

□

Now we state some simple simulation lemmas for pure terms and specific rules.

Lemma 3.5.8 (simulation of φ_k) *For every $a \in \Lambda_{dB}$ and $k \geq 0$, $\varphi'_k(a) \xrightarrow{*}_{\emptyset_S} \varphi_k(a)$.*

PROOF: By induction on a.

- if $a = m$ then clearly $\varphi'_k(m) \xrightarrow{*}_{\varphi var} \varphi_k(m)$ in all cases
- if $a = \lambda b$ then $\varphi'_k(\lambda b) \to_{\emptyset_S} \lambda\varphi'_{k+1}(b) \xrightarrow{*}^{IH}_{\emptyset_S} \lambda\varphi_{k+1}(b) = \varphi_k(\lambda b)$
- if $a = bc$ then $\varphi'_k(bc) \to_{\emptyset_S} \varphi'_k(b)\varphi'_k(c) \xrightarrow{*}^{IH}_{\emptyset_S} \varphi_k(b)\varphi_k(c) = \varphi_k(bc)$

□

Lemma 3.5.9 (simulation of e_j) *For every $a \in \Lambda_{dB}$ and $j \geq 1$, $e'_j(a) \xrightarrow{*}_{\emptyset_S} e_j(a)$.*

PROOF: By induction on a, very similar to Lemma 3.5.8. □

Then the main simulation follows:

Proposition 3.5.10 *For all $a, b \in \Lambda_{dB}$, if $a \xrightarrow{*}_{\lambda_{\emptyset dB}} b$ then $a \xrightarrow{*}_{\lambda_{\emptyset S}} b$.*

PROOF: We prove that if $a \to_{\lambda_{\emptyset dB}} b$ then $a \xrightarrow{*}_{\lambda_{\emptyset S}} b$, by induction on the position of the redex. Then the result will follow by induction on the length of the $\xrightarrow{*}_{\lambda_{\emptyset dB}}$-derivation.

1. if the reduction takes place at the root, we have the following cases:
   (a) $(\lambda 1)c \to_{dB\lambda var1} c$, then trivially $(\lambda 1)c \to_{s\lambda var1} c$
   (b) $(\lambda(m+1))c \to_{dB\lambda var2} m$, then trivially $(\lambda(m+1))c \to_{s\lambda var2} m$
   (c) $(\lambda(\lambda a))c \to_{dB\lambda\lambda} \lambda(\lambda e_1(a))\varphi_0(c)$, then
   $(\lambda(\lambda a))c \to_{s\lambda\lambda} \lambda(\lambda e'_1(a))\varphi'_0(c) \xrightarrow{*}_{\lambda_{\emptyset S}} \lambda(\lambda e_1(a))\varphi_0(c)$ by Lemmas 3.5.8 and 3.5.9
   (d) $(\lambda ab)c \to_{db\lambda app} (\lambda a)c((\lambda b)c)$, then trivially $(\lambda ab)c \to_{s\lambda app} (\lambda a)c((\lambda b)c)$
2. if the reduction is internal, the result follows easily by IH

□

Recall that for $a \in \Lambda_{dB}$, $\varphi_k(a) \in \Lambda_{dB}$ for all $k \geq 0$ and $e_j(a) \in \Lambda_{dB}$ for all $j \geq 1$. We still need the following lemmata to prepare the territory for Projection.

Lemma 3.5.11 ($(\emptyset_S, \varphi'_k)$-interchange) *For every $a \in \Lambda_{dBs}$ and $k \geq 0$, $\emptyset_S(\varphi'_k(a)) = \varphi_k(\emptyset_S(a))$.*

PROOF: $\varphi'_k(a) \xrightarrow{*}_{\emptyset S} \varphi'_k(\emptyset_S(a)) \xrightarrow{*}_{\emptyset S} \varphi_k(\emptyset_S(a))$ (by Lemma 3.5.8).
The last term is in Λ_{dB} hence an $\emptyset_S$-normal form, therefore $\emptyset_S(\varphi'_k(a)) = \varphi_k(\emptyset_S(a))$. □

Lemma 3.5.12 ($(\emptyset_S, e'_j)$-interchange) *For every $a \in \Lambda_{dBs}$ and $j \geq 1$, $\emptyset_S(e'_j(a)) = e_j(\emptyset_S(a))$.*

PROOF: Similar to Lemma 3.5.11. $e'_j(a) \xrightarrow{*}_{\emptyset S} e'_j(\emptyset_S(a)) \xrightarrow{*}_{\emptyset S} e_j(\emptyset_S(a))$ (by Lemma 3.5.9).
Again the last term is in Λ_{dB} hence an $\emptyset_S$-normal form, therefore $\emptyset_S(e'_j(a)) = e_j(\emptyset_S(a))$. □

The following four lemmas, which will be used later on, are valid for pure terms.

Lemma 3.5.13 ((φ, φ)-interchange) *For every $a \in \Lambda_{dB}$ and $k \geq 0$, $\varphi_0(\varphi_k(a)) = \varphi_{k+1}(\varphi_0(a))$.*

PROOF: $\varphi_i(\varphi_{k+i}(a)) = \varphi_{k+i+1}(\varphi_i(a))$ for all $i \geq 0$ is proved by induction on a. □

Lemma 3.5.14 ((φ, e)-interchange) *For every $a \in \Lambda_{dB}$ and $j \geq 1$, $\varphi_0(e_j(a)) = e_{j+1}(\varphi_0(a))$.*

PROOF: $\varphi_i(e_{j+i}(a)) = e_{j+i+1}(\varphi_i(a))$ for all $i \geq 0$ is proved by induction on a. □

Lemma 3.5.15 ((e, φ)-interchange) *For every $a \in \Lambda_{dB}$ and $k \geq 0$, $e_1(\varphi_{k+2}(a)) = \varphi_{k+2}(e_1(a))$.*

PROOF: $e_{i+1}(\varphi_{k+i+2}(a)) = \varphi_{k+i+2}(e_{i+1}(a))$ for all $i \geq 0$ is proved by induction on a. □

Lemma 3.5.16 ((e,e)-interchange) *For every $a \in \Lambda_{dB}$ and $j \geq 1$, $e_1(e_{j+2}(a)) = e_{j+2}(e_1(a))$.*

PROOF: $e_{i+1}(e_{j+i+2}(a)) = e_{j+i+2}(e_{i+1}(a))$ for all $i \geq 0$ is proved by induction on a. □

The proofs of the last four lemmas are by a routine induction on a. We need the results for $i = 0$, but since both $\varphi(\bullet)$ and $e(\bullet)$ increase their indices while traversing the λ binder, those general statements are needed for the induction to work.

Anyway they may admit intuition. For example, in Lemma 3.5.16, since $j \geq 1$, 1 and $j+2$ differ in 2 or more, therefore at the index level both exchanges e_1 and e_{j+2} do not overlap with each other, i.e. one is not affected by the effect of the other (in a practical setting "they might be executed in parallel").

Lemma 3.5.17 (Reduction under φ_k and under e_j) *Let $a, b \in \Lambda_{dB}$.*

1. *Let $k \geq 0$. If $a \to_{\lambda_{\emptyset dB}} b$ then $\varphi_k(a) \overset{*}{\to}_{\lambda_{\emptyset dB}} \varphi_k(b)$.*
2. *Let $j \geq 1$. If $a \to_{\lambda_{\emptyset dB}} b$ then $e_j(a) \overset{*}{\to}_{\lambda_{\emptyset dB}} e_j(b)$.*

PROOF:

1. By induction on the position of the redex. If the reduction is at the root then we have the following cases:
   - for the reduction $(\lambda 1)c \to_{\lambda_{\emptyset dB}} c$, $\varphi_k((\lambda 1)c) = (\lambda\varphi_{k+1}(1))\varphi_k(c) = (\lambda 1)\varphi_k(c)$ $\to_{\lambda_{\emptyset dB}} \varphi_k(c)$.
   - for the reduction $(\lambda(m+1))c \to_{\lambda_{\emptyset dB}} m$, $\varphi_k((\lambda(m+1))c) = (\lambda\varphi_{k+1}(m+1))\varphi_k(c)$. In case $m \leq k$, the latter equals $(\lambda(m+1))\varphi_k(c) \to_{\lambda_{\emptyset dB}} m = \varphi_k(m)$. Else, $m > k$ and the term in the right equals $(\lambda(m+2))\varphi_k(c) \to_{\lambda_{\emptyset dB}} m+1 = \varphi_k(m)$.
   - for the reduction $(\lambda cd)e \to_{\lambda_{\emptyset dB}} (\lambda c)e((\lambda d)e)$, $\varphi_k((\lambda cd)e) = (\lambda\varphi_{k+1}(c)\varphi_{k+1}(d))\varphi_k(e)$ $\to_{\lambda_{\emptyset dB}} (\lambda\varphi_{k+1}(c))\varphi_k(e)((\lambda\varphi_{k+1}(d))\varphi_k(e)) = \varphi_k((\lambda c)e((\lambda d)e))$.
   - for the reduction $(\lambda\lambda c)d \to_{\lambda_{\emptyset dB}} \lambda(\lambda e_1(c))\varphi_0(d)$, $\varphi_k((\lambda\lambda c)d) = (\lambda\lambda\varphi_{k+2}(c))\varphi_k(d)$ $\to_{\lambda_{\emptyset dB}} \lambda(\lambda e_1(\varphi_{k+2}(c)))\varphi_0(\varphi_k(d))$ $= \lambda(\lambda\varphi_{k+2}(e_1(c)))\varphi_{k+1}(\varphi_0(d))$ (by Lemmas 3.5.15 and 3.5.13) $= \varphi_k(\lambda(\lambda(e_1(c))\varphi_0(d)))$.

   If the reduction is internal, it is straightforward.
2. By induction on a. If the reduction is at the root then we have the following cases:
   - for the reduction $(\lambda 1)c \to_{\lambda_{\emptyset dB}} c$, $e_j((\lambda 1)c) = (\lambda e_{j+1}(1))e_j(c) = (\lambda 1)e_j(c) \to_{\lambda_{\emptyset dB}} e_j(c)$.

- for the reduction $(\lambda(m+1))c \to_{\lambda_{\emptyset dB}} m$, $e_j((\lambda(m+1))c) = (\lambda e_{j+1}(m+1))e_j(c)$.
  In case $m < j$, the latter equals $(\lambda(m+1))e_j(c) \to_{\lambda_{\emptyset dB}} m = e_j(m)$.
  Else if $m = j$, the term in the right equals $(\lambda(m+2))e_j(c) \to_{\lambda_{\emptyset dB}} m+1 = e_j(m)$.
  Else if $m = j+1$ (thus $m \geq 2$), the term in the right equals $(\lambda e_m(m+1))e_j(c)$
  $= (\lambda m)e_j(c) \to_{\lambda_{\emptyset dB}} m-1 = e_j(m)$ since $j = m-1$.
  Else $m > j+1$ and the term in the right equals $(\lambda(m+1))e_j(c) \to_{\lambda_{\emptyset dB}} m = e_j(m)$.
- for the reduction $(\lambda cd)e \to_{\lambda_{\emptyset dB}} (\lambda c)e((\lambda d)e)$, $e_j((\lambda cd)e) = (\lambda e_{j+1}(c)e_{j+1}(d))e_j(e)$
  $\to_{\lambda_{\emptyset dB}} (\lambda e_{j+1}(c))e_j(e)((\lambda e_{j+1}(d))e_j(e)) = e_j((\lambda c)e((\lambda d)e))$.
- for the reduction $(\lambda\lambda c)d \to_{\lambda_{\emptyset dB}} \lambda(\lambda e_1(c))\varphi_0(d)$, $e_j((\lambda\lambda c)d) = (\lambda\lambda e_{j+2}(c))e_j(d)$
  $\to_{\lambda_{\emptyset dB}} \lambda(\lambda e_1(e_{j+2}(c)))\varphi_0(e_j(d))$
  $= \lambda(\lambda e_{j+2}(e_1(c)))e_{j+1}(\varphi_0(d))$ (by Lemmas 3.5.16 and 3.5.14)
  $= e_j(\lambda(\lambda(e_1(c))\varphi_0(d)))$.

If the reduction is internal, it is straightforward.

□

Then the Projection Lemma comes out.

Proposition 3.5.18 (Projection Lemma) *Let $a, b \in \Lambda_{dBs}$.*
If $a \xrightarrow{*}_{\lambda_{\emptyset S}} b$ then $\emptyset_S(a) \xrightarrow{*}_{\lambda_{\emptyset dB}} \emptyset_S(b)$.

Proof: By induction on the position of the redex.

If the reduction is at the root, the interesting case is:
$(\lambda(\lambda c))d \to_{s\lambda\lambda} \lambda(\lambda e'_1(c))\varphi'_0(d)$, then, by Lemma 3.5.7,
$\emptyset_S((\lambda(\lambda c))d) = (\lambda(\lambda\emptyset_S(c)))\emptyset_S(d) \to_{dB\lambda\lambda} \lambda(\lambda e_1(\emptyset_S(c)))\varphi_0(\emptyset_S(d))$
$= \lambda(\lambda\emptyset_S(e'_1(c)))\emptyset_S(\varphi'_0(d))$ (by Lemmas 3.5.11 and 3.5.12)
$= \emptyset_S(\lambda(\lambda e'_1(c))\varphi'_0(d))$ (by Lemma 3.5.7).

If the reduction is internal in $cd \to_{\lambda_{\emptyset S}} c'd$, $cd \to_{\lambda_{\emptyset S}} cd'$ or $\lambda c \to_{\lambda_{\emptyset S}} \lambda c'$, use Lemma 3.5.7 (1) and (2).

If the reduction is internal in $\varphi'_k(c) \to_{\lambda_{\emptyset S}} \varphi'_k(c')$, then $\emptyset_S(\varphi'_k(c)) = \varphi_k(\emptyset_S(c))$ (by Lemma 3.5.11)
$\xrightarrow{*}_{\lambda_{\emptyset dB}} \varphi_k(\emptyset_S(c'))$ (using the IH and Lemma 3.5.17 (1))
$= \emptyset_S(\varphi'_k(c'))$ (by Lemma 3.5.11 again).

If the reduction is internal in $e'_j(c) \to_{\lambda_{\emptyset S}} e'_j(c')$, then analogously
$\emptyset_S(e'_j(c)) = e_j(\emptyset_S(c))$ (by Lemma 3.5.12)
$\xrightarrow{*}_{\lambda_{\emptyset dB}} e_j(\emptyset_S(c'))$ (using the IH and Lemma 3.5.17 (2))
$= \emptyset_S(e'_j(c'))$ (by Lemma 3.5.12 again). □

Corollary 3.5.19 (Confluence of $\lambda_{\emptyset S}$) *Let $a, b, c \in \Lambda_{dBs}$. If $a \stackrel{*}{\to}_{\lambda_{\emptyset S}} b$ and $a \stackrel{*}{\to}_{\lambda_{\emptyset S}} c$ then there exists $d \in \Lambda_{dBs}$ such that $b \stackrel{*}{\to}_{\lambda_{\emptyset S}} d$ and $c \stackrel{*}{\to}_{\lambda_{\emptyset S}} d$.*

PROOF: Suppose $a \stackrel{*}{\to}_{\lambda_{\emptyset S}} b$ and $a \stackrel{*}{\to}_{\lambda_{\emptyset S}} c$. Then by Proposition 3.5.18 $\emptyset_S(a) \stackrel{*}{\to}_{\lambda_{\emptyset dB}} \emptyset_S(b)$ and $\emptyset_S(a) \stackrel{*}{\to}_{\lambda_{\emptyset dB}} \emptyset_S(c)$. By the confluence of $\lambda_{\emptyset dB}$ (Corollary 3.4.1), there exists $d \in \Lambda_{dB}$ such that $\emptyset_S(b) \stackrel{*}{\to}_{\lambda_{\emptyset dB}} d$ and $\emptyset_S(c) \stackrel{*}{\to}_{\lambda_{\emptyset dB}} d$. By Simulation (Proposition 3.5.10), $\emptyset_S(b) \stackrel{*}{\to}_{\lambda_{\emptyset S}} d$ and $\emptyset_S(c) \stackrel{*}{\to}_{\lambda_{\emptyset S}} d$. Using this, and that $b \stackrel{*}{\to}_{\emptyset_S} \emptyset_S(b)$ and $c \stackrel{*}{\to}_{\emptyset_S} \emptyset_S(c)$, we are done. □

It turns out that $\lambda_{\emptyset S}$ is sound with respect to $\lambda_{\emptyset dB}$ (it is a consequence of the Projection Lemma), and like the latter, it is relatively sound with respect to β_{dB}. In brief

Corollary 3.5.20 *If $a, b \in \Lambda_{dB}$ and $a \stackrel{*}{\to}_{\lambda_{\emptyset S}} b$ then:*

1. *$a \stackrel{*}{\to}_{\lambda_{\emptyset dB}} b$, and (therefore) $a =_{\beta_{dB}} b$*
2. *if $b \in NF_{\lambda_{\emptyset dB}}$ then $a \stackrel{*}{\to}_{\beta_{dB}} b$.*

PROOF:

1. Immediate using Proposition 3.5.18 and Remark 3.5.5.
2. Immediate by item (1) and CR of β_{dB}.

□

3.5.1 Typed $\lambda_{\emptyset S}$

Types and environments are defined for all the de Bruijn calculi discussed in this chapter, as follows. We deal with sequent-based typing, and recall the syntax and typing rules for the simply typed λ-calculus in de Bruijn notation. The types we will treat are generated as first order terms from a set of basic types T with the binary type operator $\to$. Environments will be lists of types.

The system **L1** (see the preliminaries) is also applicable to $\lambda_{\emptyset dB}$, where good results follow. We will not focus on this system, but will go directly to simply typed $\lambda_{\emptyset S}$. Therefore we add below the rules to handle types for the new terms of $\lambda_{\emptyset S}$ in a Curry style as follows. Remark that we are not marking the abstractions with types as it is done in other syntactical variations of typing systems (which will be used in another chapter).

Definition 3.5.21 *The typing system* **LS** *is defined with the same set of types, and as typing rules the set of* **L1** *rules of simply-typed classical λ-calculus plus the following two typing rules:*

$$(\mathbf{LS}-\varphi)\qquad \frac{A_1,\ldots,A_n \vdash a : B}{A_1,\ldots,A_k,A,A_{k+1},\ldots,A_n \vdash \varphi'_k(a) : B} \qquad n \geq k \geq 0$$

$$(\mathbf{LS}-e)\qquad \frac{A_1,\ldots,A_n \vdash a : B}{A_1,\ldots,A_{j-2},A_{j-1},A_{j+1},A_j,A_{j+2},A_{j+3}\ldots,A_n \vdash e'_j(a) : B} \qquad n \geq j \geq 1$$

From the logical point of view, the typing rule $(\mathbf{LS}-\varphi)$ represents a *weakening* rule, since it allows the addition of an extra type in an environment (associated to an extra hypothesis, but in the proper place). And the typing rule $(\mathbf{LS}-e)$ represents an *exchange* rule, since it allows the swap between two consecutive types in an environment.

We will say a term $a \in \Lambda_{dBs}$ is typed, or typable (in **LS**) iff there exist an environment Γ and a type A such that $\Gamma \vdash_{\mathbf{LS}} a : A$.

The following expected results tell about the relation between both typing systems, type preservation and normalization of typable terms.

Lemma 3.5.22 (Type preservation) *Let $a \in \Lambda_{dB}$. Then $\Gamma \vdash_{\mathbf{L1}} a : A$ iff $\Gamma \vdash_{\mathbf{LS}} a : A$.*

Proof: Can be done by a simple induction on a. □

This typing system admits a Generation Lemma for terms of all forms: m, ab, λa, $\varphi'_k(a)$ and $e'_j(a)$. (See (12) for these results in the classical setting.)

Lemma 3.5.23 (Generation Lemma for $\lambda_{\emptyset S}$)

1. *Let $n \geq 0, m \geq 1$. If $A_1,\ldots,A_n \vdash m : B$, then $m \leq n$ and $B = A_m$.*
2. *If $A_1,\ldots,A_n \vdash ab : B$, then there exists a type A such that $A_1,\ldots,A_n \vdash a : A \to B$ and $A_1,\ldots,A_n \vdash b : A$.*
3. *If $A_1,\ldots,A_n \vdash \lambda a : C$, then there exist types A and B such that $C = A \to B$ and $A, A_1,\ldots,A_n \vdash a : B$.*
4. *Let $n \geq k \geq 0$. If $A_1,\ldots,A_k,A,A_{k+1},\ldots,A_n \vdash \varphi'_k(a) : B$, then $A_1,\ldots,A_n \vdash a : B$.*
5. *Let $n \geq j \geq 1$. If $A_1,\ldots,A_{j-2},A_{j-1},A_{j+1},A_j,A_{j+2},A_{j+3}\ldots,A_n \vdash e'_j(a) : B$, then $A_1,\ldots,A_n \vdash a : B$.*

Proof: All the items are proved by induction on the typing derivation. (The proofs of items (1), (2) and (3) are analogous to the classical ones.) □

Proposition 3.5.24 (Subject reduction) *Let $a, b \in \Lambda_{dBs}$. If $a \xrightarrow{*}_{\lambda_{\emptyset S}} b$ and $\Gamma \vdash_{\mathbf{LS}} a : A$, then $\Gamma \vdash_{\mathbf{LS}} b : A$.*

PROOF: If $a \rightarrow_{\lambda_{\emptyset S}} b$ and $\Gamma \vdash_{\mathbf{LS}} a : A$, then $\Gamma \vdash_{\mathbf{LS}} b : A$ can be proved by induction on a. We illustrate with the following cases:

1. root reduction of the rule $\varphi'_k(m) \rightarrow_{\varphi var} m$ where $m \leq k$. We have:

$$A_1, \ldots, A_k, A, A_{k+1}, \ldots, A_n \vdash \varphi'_k(m) : B$$

where $n \geq k \geq 0$. By generation, $A_1, \ldots, A_n \vdash m : B$. Again by generation, $B = A_m$. Then, since $m \leq k$, $A_1, \ldots, A_k, A, A_{k+1}, \ldots, A_n \vdash m : A_m$ and we are done.

2. root reduction of the rule $\varphi'_k(m) \rightarrow_{\varphi var} m + 1$ where $m > k$. We have:

$$A_1, \ldots, A_k, A, A_{k+1}, \ldots, A_n \vdash \varphi'_k(m) : B$$

where $n \geq k \geq 0$. By generation, $A_1, \ldots, A_n \vdash m : B$. Again by generation, $B = A_m$. Then, since $m > k$, $A_1, \ldots, A_k, A, A_{k+1}, \ldots, A_n \vdash m + 1 : A_m$ and we are done.

3. root reduction of the rule $e'_j(m) \rightarrow_{evar} m$ where $m < j$. We have by generation the inference step:

$$\frac{A_1, \ldots, A_n \vdash m : B}{A_1, \ldots, A_{j-2}, A_{j-1}, A_{j+1}, A_j, A_{j+2}, A_{j+3} \ldots, A_n \vdash e'_j(m) : B}$$

where $n \geq j \geq 1$. By generation on the premise, $B = A_m$. Then, since $m < k$, we have $A_1, \ldots, A_{j-2}, A_{j-1}, A_{j+1}, A_j, A_{j+2}, A_{j+3} \ldots, A_n \vdash m : A_m$ and we are done.

4. root reduction of the rule $e'_j(j) \rightarrow_{evar} j + 1$. We have by generation the inference step:

$$\frac{A_1, \ldots, A_{j-2}, A_{j-1}, A_j, A_{j+1}, A_{j+2}, \ldots, A_n \vdash j : B}{A_1, \ldots, A_{j-2}, A_{j-1}, A_{j+1}, A_j, A_{j+2}, A_{j+3} \ldots, A_n \vdash e'_j(j) : B}$$

where $n \geq j \geq 1$. By generation on the premise, $B = A_j$. Then we have $A_1, \ldots, A_{j-2}, A_{j-1}, A_{j+1}, A_j, A_{j+2}, A_{j+3} \ldots, A_n \vdash j + 1 : A_j$ and we are done.

5. root reduction of the rule $e'_j(j + 1) \rightarrow_{evar} j$. We have by generation the inference step:

$$\frac{A_1, \ldots, A_{j-2}, A_{j-1}, A_j, A_{j+1}, A_{j+2}, \ldots, A_n \vdash j + 1 : B}{A_1, \ldots, A_{j-2}, A_{j-1}, A_{j+1}, A_j, A_{j+2}, A_{j+3} \ldots, A_n \vdash e'_j(j + 1) : B}$$

where $n \geq j \geq 1$. By generation on the premise, $B = A_{j+1}$. Then we have $A_1, \ldots, A_{j-2}, A_{j-1}, A_{j+1}, A_j, A_{j+2}, A_{j+3} \ldots, A_n \vdash j : A_{j+1}$ and we are done.

6. root reduction of the rule $e'_j(m) \rightarrow_{evar} m$ where $m > j + 1$. We have by generation the inference step:

$$\frac{A_1, \ldots, A_{j-2}, A_{j-1}, A_j, A_{j+1}, A_{j+2}, \ldots, A_n \vdash m : B}{A_1, \ldots, A_{j-2}, A_{j-1}, A_{j+1}, A_j, A_{j+2}, A_{j+3} \ldots, A_n \vdash e'_j(m) : B}$$

where $n \geq j \geq 1$. By generation on the premise, $B = A_m$. Then we have $A_1, \ldots, A_{j-2}, A_{j-1}, A_{j+1}, A_j, A_{j+2}, A_{j+3} \ldots, A_n \vdash m : A_m$ and we are done.

For the root reductions of the other $\lambda_{\emptyset dB}$-rules the Generation Lemma for the typing rules (**LS**-*var*), (**L1**-*varn*), (**L1**-*abs*) and (**L1**-*app*) should be used and the proof proceeds. The other 2 rules and the 5 inductive cases present no problem at all. □

The most that can be stated about typed terms and normalization is given by the

Proposition 3.5.25 (Weak normalization of simply typed $\lambda_{\emptyset S}$) *Let $a \in \Lambda_{dBs}$. If a is typable then a is WN (but may not be SN).*

Proof: Suppose $\Gamma \vdash_{\mathbf{LS}} a : A$. Since $a \xrightarrow{*}_{\emptyset S} \emptyset_S(a)$ then by Proposition 3.5.24 $\Gamma \vdash_{\mathbf{LS}} \emptyset_S(a) : A$. Since $\emptyset_S(a)$ is pure, by Lemma 3.5.22 $\emptyset_S(a)$ is SN in classical λ-calculus à la de Bruijn (11; 12). Then, by Corollary 3.4.10 (2) and Simulation (Proposition 3.5.10),
$\emptyset_S(a) \in WN_{\lambda_{\emptyset S}}$ thus $a \in WN_{\lambda_{\emptyset S}}$. □

3.6 Relating the calculi

This section deals with the relationship between the various calculi we have treated in this chapter, from the point of view of soundness, simulation and PSN.

We have a calculus over names, with a version over indices, satisfying simulation of β-reduction, confluence, relative soundness but not PSN, and which is somehow easier to implement than other explicit substitution calculi since it is entirely constructed over the pure λ-terms (no need of closures) and with only four rules.

Yet we "conjecture" that $\lambda_\emptyset$ is the "simplest" of all λ-calculi with names and (in some manner) explicit substitution. Reasons supporting this assertion are the following. We have that:

1. the set of $\lambda_\emptyset$-terms is Λ, the set of $\lambda_{\emptyset dB}$-terms is Λ_{dB}
2. a β-step (β_{dB}-step) may become several $\lambda_\emptyset$-steps ($\lambda_{\emptyset dB}$-steps) over the β-equivalence (β_{dB}-equivalence) class
3. every WN-term preserves its unique normal form, in $\lambda_\emptyset$ and $\lambda_{\emptyset dB}$
4. $NF_{\lambda_\emptyset} = NF_\beta$, $NF_{\lambda_{\emptyset dB}} = NF_{\beta dB}$
5. $WN_{\lambda_\emptyset} = WN_\beta$, $WN_{\lambda_{\emptyset dB}} = WN_{\beta dB}$
6. $SN_{\lambda_\emptyset} \subset SN_\beta$, $SN_{\lambda_{\emptyset dB}} \subset SN_{\beta dB}$
7. $\lambda_\emptyset$ ($\lambda_{\emptyset dB}$) has a simply typed version in which typed terms are WN and preserve λ (λ_{dB}) simple typing

And other relevant assertions can be stated for the $\lambda_{\emptyset S}$ de Bruijn version as we have illustrated.

Being in the presence of various calculi, we will relate the systems by expressing them in a general way. We give the following general (almost "informal")

Definition 3.6.1 *Given two calculi λ_1 and λ_2 (that is, two higher-order rewriting systems which are variants of λ-calculus in some "clear" way), an appropriate mapping between their respective sets of terms is an effective bi-injective pair of functions (or algorithms), that is a computable way to take a term from one of them and obtain a unique term from the other one.*

Definition 3.6.2 *Given two calculi λ_1 and λ_2 (in the above sense) and an appropriate pair of mappings (w, u) between them, we will write:*

- *$So(\lambda_1, \lambda_2)$ iff λ_1 is sound with respect to λ_2 via the mapping, i.e. for every pair of λ_1-terms a, b, if $a \xrightarrow{*}_{\lambda_1} b$ then $w(a) \xrightarrow{*}_{\lambda_2} w(b)$*
- *$Rs(\lambda_1, \lambda_2)$ iff λ_1 is relatively sound with respect to λ_2 via the mapping, i.e. for every pair of λ_1-terms a, b, if $a \xrightarrow{*}_{\lambda_1} b$ then, $w(a) =_{\lambda_2} w(b)$ and $w(a) \xrightarrow{*}_{\lambda_2} w(b)$ if b is a λ_1-normal form*
- *$Si(\lambda_1, \lambda_2)$ iff λ_1 simulates λ_2 via the mapping, i.e. for every pair of λ_2-terms a, b, if $a \rightarrow_{\lambda_2} b$ then $u(a) \xrightarrow{+}_{\lambda_1} u(b)$*
- *$PSN(\lambda_1, \lambda_2)$ iff λ_1 is PSN with respect to λ_2 via the mapping, i.e. for every term a of λ_1, $a \in SN_{\lambda_1}$ implies $w(a) \in SN_{\lambda_2}$*

All definitions are in the sense given at the beginning of the chapter.

As a generalization of some of the results in the previous section, we have

Proposition 3.6.3 *Given the calculi λ_1, λ_2 and λ_3 the following statements hold (where appropriate mappings are to be considered):*

1. *$So(\lambda_1, \lambda_1)$*
2. *$Rs(\lambda_1, \lambda_1)$*
3. *$Si(\lambda_1, \lambda_1)$*
4. *$PSN(\lambda_1, \lambda_1)$*
5. *$So(\lambda_1, \lambda_2) \Rightarrow Rs(\lambda_1, \lambda_2)$*
6. *$So(\lambda_1, \lambda_2), So(\lambda_2, \lambda_3) \Rightarrow So(\lambda_1, \lambda_3)$*
7. *$So(\lambda_1, \lambda_2), Rs(\lambda_2, \lambda_3) \Rightarrow Rs(\lambda_1, \lambda_3)$*

8. $Rs(\lambda_1, \lambda_2), Rs(\lambda_2, \lambda_3) \Rightarrow Rs(\lambda_1, \lambda_3)$

9. $Si(\lambda_1, \lambda_2), Si(\lambda_2, \lambda_3) \Rightarrow Si(\lambda_1, \lambda_3)$

10. $PSN(\lambda_1, \lambda_2), PSN(\lambda_2, \lambda_3) \Rightarrow PSN(\lambda_1, \lambda_3)$

PROOF: All proofs are nearly immediate once fixed the subsets of terms and the mappings. □

There are trivial examples where these "relations" hold, as when relating a calculus with a sub-calculus, for example when a new rule is added which is simulated by many-steps of an already present rule (or set of rules).

Example 3.6.4 *Applying the previous proposition to $\lambda_{\emptyset S}$, $\lambda_{\emptyset dB}$, β_{dB} and β (classical λ-calculus), we have for instance the following inferences:*

1. $So(\beta_{dB}, \beta)$ *(classical isomorphism)*

2. $Rs(\beta_{dB}, \beta)$ *(from 1)*

3. $So(\lambda_{\emptyset S}, \lambda_{\emptyset dB})$ *(Corollary 3.5.20)*

4. $Rs(\lambda_{\emptyset S}, \lambda_{\emptyset dB})$ *(from 3)*

5. $Rs(\lambda_{\emptyset dB}, \beta_{dB})$ *(Lemma 3.4.5)*

6. $Rs(\lambda_{\emptyset S}, \beta_{dB})$ *(from 4, 5)*

7. $Rs(\lambda_{\emptyset S}, \beta)$ *(from 6, 2)*

3.7 Conclusion and future work

We have studied a λ-calculus with names using the set of pure λ-terms, without closure operators, which in some sense embeds $\lambda\mathtt{x}$ but with only four rules, and no substitutions. We have in this way proved the relative soundness in the sense that only correct calculations with respect to β-equivalence and normal forms can be done. We have seen that a β-step may become several $\lambda_\emptyset$-steps over the β-equivalence class. This means that the new terms which may appear within a derivation simulating a classical β-reduction are just expansions of the classical β-reduct, thus β-equivalence is preserved. This also indicates that the differences of this calculus with classical λ-calculus are non-trivial, in the sense that new terms would appear along the derivations with respect to the latter. We have introduced two de Bruijn versions of this calculus, preserving the same properties.

It might be viewed that the closure operators become necessary from the fact that the proposed variant of λ-calculus is not PSN, although as we stated before this was not the

historical motivation. Closure operators have appeared in the literature motivating the explicit substitution paradigm of λ-calculus, as a two-phase procedure for calculation. On one hand, the λ-abstraction (along with the application), and on the other, the closure. Without this distinction, one loses PSN as well as classical soundness but it preserves β-equivalence classes and correctly calculates normal forms. The other main requirements work well and this calculus is suitable for implementation.

Since $\lambda_{\emptyset}$ is already non-PSN, the addition of a possible composition rule would be plausible. The addition of η-reduction may also be viable: for $\lambda_{\emptyset dB}$ it would read $\lambda(\varphi_0(a)1) \to a$. Also a version of $\lambda_{\emptyset}$ with constructors (see chapter 7) could be an interesting topic to study for comparison with other pattern-based λ-calculi.

It would be interesting to study the possible embedding of other de Bruijn calculi onto $\lambda_{\emptyset dB}$. Also to study other richer typing systems for these calculi.

As a future task it remains to study the confluence on open terms for possible extensions of $\lambda_{\emptyset S}$, which will require to transform the exchange lemmas 3.5.13, 3.5.14, 3.5.15, 3.5.16 into rules by giving them some appropriate orientation. Another interesting facet is the identification of (more) non-trivial chains of calculi enjoying step-wise simulation, soundness and, as a particular aspect, to study this relative PSN of one calculus with respect to another.

3.8 Appendix. A garbage-collection rule for $\lambda_{\emptyset dB}$

In this short appendix we propose a variant of $\lambda_{\emptyset dB}$ in which following the idea of the full (gc)-rule in $\lambda\mathtt{x}$, the $db\lambda var2$-rule is replaced by a garbage-collection rule.

For this purpose we extend the updating functions $U_k^i(\bullet)$ by including the case $i = 0$, to be named V_k^i:

$$\begin{array}{lcl} V_k^i(m) & = & \begin{cases} m-1 & \text{if } m > k \text{ and } i = 0 \\ m+i-1 & \text{if } m > k \text{ and } i > 0 \\ m & \text{if } m \leq k \end{cases} \\ V_k^i(ab) & = & V_k^i(a)V_k^i(b) \\ V_k^i(\lambda a) & = & \lambda V_{k+1}^i(a) \end{array}$$

With the syntax of (pure) de Bruijn terms, we consider the following rules (recall the definition of $FV(\bullet)$ in the preliminaries):

$$\begin{array}{llll} (\lambda 1)a & \to_{db\lambda var1} & a & \\ (\lambda a)b & \to_{db\lambda gc} & V_0^0(a) & \text{if } 1 \notin FV(a) \\ (\lambda ab)c & \to_{db\lambda app} & (\lambda a)b((\lambda a)c) & \\ (\lambda(\lambda a))c & \to_{db\lambda\lambda} & \lambda(\lambda e_1(a))V_0^2(b) & \end{array}$$

where $V_k^i(\bullet)$ is the above updating operator, and $e_k(\bullet)$ is the $(k, k+1)$-exchange operator. The goal of $V_0^0(a)$ is to decrement every free index of a by 1. (Actually we do not need to define $V_k^i(\bullet)$ for other values of $i \neq 0, 2$ in this formulation.)

We will call $\lambda_{\emptyset dBg}$ the preceding calculus over de Bruijn terms.

Lemma 3.8.1 *If* $1 \in FV(e_1(a))$ *then* $2 \in FV(a)$.

PROOF: Since $e_j(e_j(a)) = a$, it suffices to show that for every $j \geq 1$, $j \in FV(a)$ implies $j+1 \in FV(e_j(a))$. It is done by induction on a. □

Lemma 3.8.2 *Let* $j \geq 1$. *If* $j \notin FV(a)$ *then* $V_j^0(a) = V_{j-1}^0(a)$.

PROOF: By induction on a. The case $a = m$ is clear since $m \neq j$. The interesting case is $a = \lambda a'$, where $j \notin FV(\lambda a') = FV(a') - 1$ so $j+1 \notin FV(a')$ and then by IH $V_{j+1}^0(a') = V_j^0(a')$, so $\lambda V_{j+1}^0(a') = \lambda V_j^0(a')$ and then $V_j^0(\lambda a') = V_{j-1}^0(\lambda a')$[1]. □

Lemma 3.8.3 *If* $j \notin FV(e_j(a))$ *then* $V_j^0(e_j(a)) = V_j^0(a)$.

PROOF: By induction on a. □

Remark 3.8.4 *If* $a \to_{\lambda_{\emptyset dB}} b$ *then* $a \to_{\lambda_{\emptyset dBg}} b$.

PROOF: Obvious since the $db\lambda var$-rule is subsumed by the $db\lambda gc$-rule. □

With respect to the converse for many steps, we need

Lemma 3.8.5 *For every de Bruijn term* a *and* $i \in \mathbb{N}$, $|e_i(a)| = |a|$.

PROOF: Easy induction on a. □

Lemma 3.8.6 states that the $db\lambda gc$-rule can be simulated in $\lambda_{\emptyset dB}$.

Lemma 3.8.6 *If* $1 \notin FV(a)$ *then* $(\lambda a)b \xrightarrow{*}_{\lambda_{\emptyset dB}} V_0^0(a)$.

PROOF: By induction on the position of the redex.

- if the reduction occurs at the root, then the following cases may occur:
  - if $a = 1$, it is vacuous since $1 \in FV(1)$
  - if $a = m+1$, clearly $(\lambda(m+1))b \to_{db\lambda var1} m = V_0^0(m+1)$
  - if $a = b_1 b_2$ then $(\lambda b_1 b_2)c \to_{db\lambda app} (\lambda b_1)c((\lambda b_2)c)$
    $\xrightarrow{*}{}^{IH}_{\lambda_{\emptyset dB}} V_0^0(b_1)V_0^0(b_2)$ (since $1 \notin FV(b_1) \cup FV(b_2) = FV(b_1 b_2)$)
    $= V_0^0(b_1 b_2)$

[1] Actually a stronger version holds: if $k < j$ and $k+1, k+2, \ldots, j \notin FV(a)$ then $V_k^0(a) = V_{k+1}^0(a) = \cdots = V_j^0(a)$.

– if $a = (\lambda a_1)$ then $(\lambda\lambda a_1)c \to_{db\lambda\lambda} d$ with $d = \lambda(\lambda e_1(a_1))V_0^2(c)$ and since $1 \notin FV(\lambda a_1)$ $2 \notin FV(a_1)$ thus $1 \notin FV(e_1(a_1))$ by Lemma 3.8.1. Lemma 3.8.5 guarantees that $|a_1| = |e_1(a_1)| < |\lambda a_1| = |a|$, then the IH can be used:
$(\lambda e_1(a_1))V_0^2(c) \stackrel{*}{\to}_{\lambda_{\emptyset dB}} V_0^0(e_1(a_1))$ since $1 \notin FV(e_1(a_1))$
$= V_1^0(e_1(a_1))$ by Lemma 3.8.2 taking $j = 1 \notin FV(e_1(a_1))$
$= V_1^0(a_1)$ by Lemma 3.8.3 using again that $1 \notin FV(e_1(a_1))$, so
$d \stackrel{*}{\to}_{\lambda_{\emptyset dB}} \lambda V_1^0(a_1) = V_0^0(a)$ and we are done.

- if the reduction is internal, it is straightforward.

□

Corollary 3.8.7 (Simulation of $\lambda_{\emptyset dBg}$) *If $a \stackrel{*}{\to}_{\lambda_{\emptyset dBg}} b$ then $a \stackrel{*}{\to}_{\lambda_{\emptyset dB}} b$.*

PROOF: We prove that if $a \to_{\lambda_{\emptyset dBg}} b$ then $a \stackrel{*}{\to}_{\lambda_{\emptyset dB}} b$ by induction on a. The internal reductions are straightforward. The $db\lambda gc$-rule is handled by Lemma 3.8.6. Then the result follows by induction on the length of the derivation. □

So both relations $\stackrel{*}{\to}_{\lambda_{\emptyset dB}}$ and $\stackrel{*}{\to}_{\lambda_{\emptyset dBg}}$ happen to coincide. This coincidence does not hold in the setting of $\lambda\mathtt{x}$ (having the (Gc)-rule) and $\lambda\mathtt{x}^-$ (having the more restrictive $(Var2)$-rule), since $\lambda\mathtt{x}$ allows derivation steps which are not simulated by $\lambda\mathtt{x}^-$. More precisely, $\lambda\mathtt{x}^-$-rules cannot simulate the (Gc)-rule since a closure may block a reduction. In $\lambda_{\emptyset dB}$ there is no such a blocking possibility since there are no closures. As immediate consequences we have the following corollaries (we omit details).

Corollary 3.8.8 (Soundness with respect to $=_{\beta_{dB}}$) *If $a \stackrel{*}{\to}_{\lambda_{\emptyset dBg}} b$ then $a =_{\beta_{dB}} b$*

Corollary 3.8.9 (Confluence of $\lambda_{\emptyset dBg}$) *If $a \stackrel{*}{\to}_{\lambda_{\emptyset dBg}} a_1$ and $a \stackrel{*}{\to}_{\lambda_{\emptyset dBg}} a_2$ then there exists a_3 such that $a_1 \stackrel{*}{\to}_{\lambda_{\emptyset dBg}} a_3$ and $a_2 \stackrel{*}{\to}_{\lambda_{\emptyset dBg}} a_3$.*

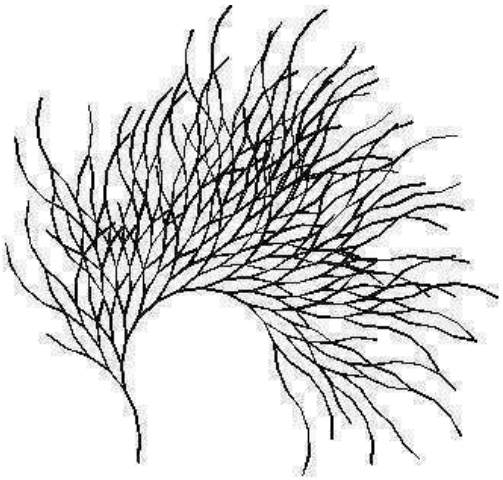
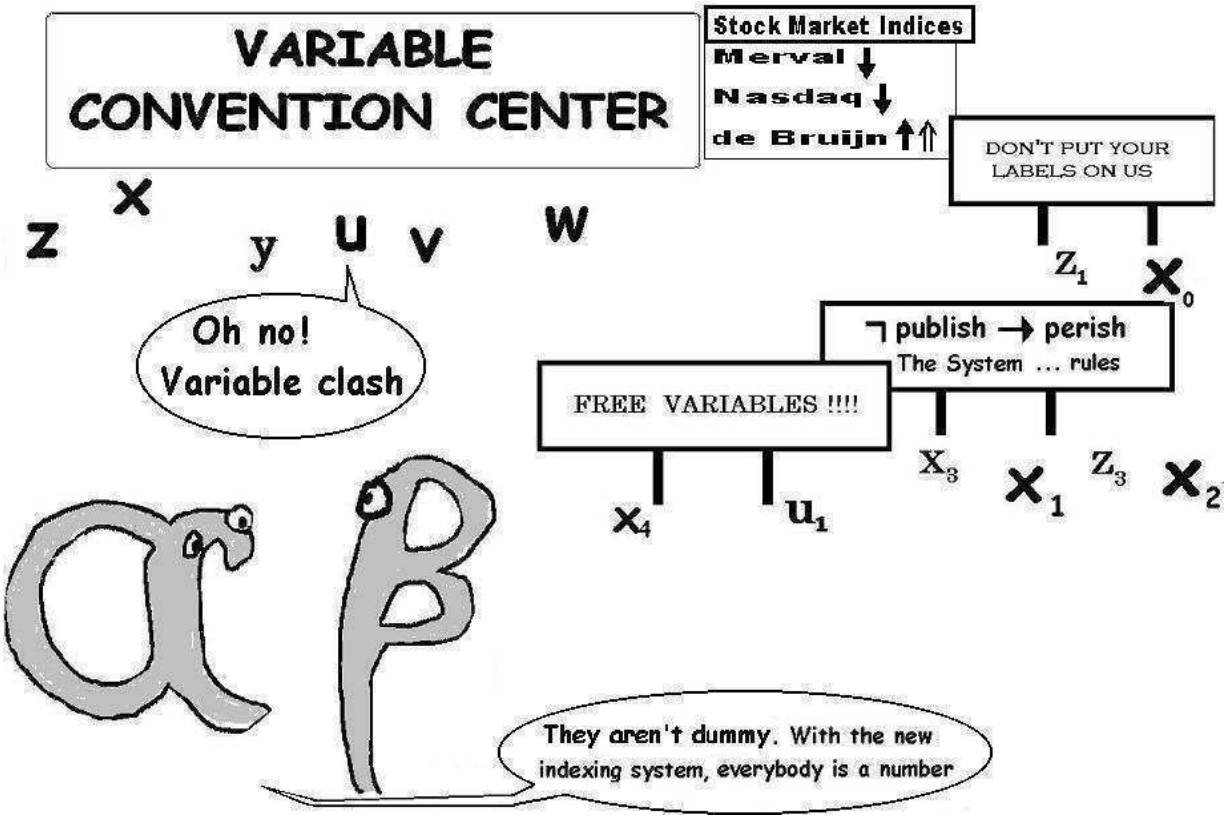


Figure 3.1: $S((S(SS))K)K(SSSSS)(S(S(SS)K)S(SKK))$ after 80 left-most steps



Chapter 4

Perpetuality and Strong Normalization in $\lambda\upsilon$

> *He that would know what kind of idea it is to which we give the name of INFINITY, cannot do it better than by considering to what infinity is by the mind more immediately attributed; and then how the mind comes to frame it.* – J. Locke

> *... by being able to repeat the idea of any length of duration we have in our minds, with all the endless addition of number, we come by the idea of ETERNITY. For we find in ourselves, we can no more come to an end of such repeated ideas than we can come to the end of number ...* – J. Locke

ABSTRACT We prove a perpetuality result for the $\lambda\upsilon$ calculus of explicit substitution, based on "safe" reductions. We give as an application a set of deterministic inference rules which characterize inductively the strongly normalizing terms and an effective perpetual reduction strategy for $\lambda\upsilon$.

4.1 Introduction

In most rewriting systems, in particular *term rewriting systems* (TRSs) (9) and explicit substitution calculi (13; 15; 35), the concept of *perpetuality* (49; 50; 51; 52; 82) can be defined and studied. This notion means, for any term, a way of "preserving" infinite derivations starting from it when they exist. When one finds that a calculus satisfies perpetuality in the given sense, then strong normalization results and perpetual rewriting strategies can be achieved. Perpetual strategies appear in a more or less straightforward way when a calculus satisfies orthogonality (14), but when this does not happen a specific analysis should be done.

The $\lambda\upsilon$- (13) and λs- (43) calculi of explicit substitution have minimal substitutions operators and rules in some sense ($/$, $\uparrow$, $\Uparrow$ for $\lambda\upsilon$ and σ, φ for λs). They can be seen as translating the

main ideas of the $\lambda\mathtt{x}$-calculus, which is a named explicit substitution calculus, into a de Bruijn setting. These systems offer the advantage that they rely on de Bruijn indices and the number of rules is relatively small compared with those from other calculi. and somehow complicated reduction rules. For that reason most of the results for $\lambda\mathtt{x}$ should be transferred to other calculi, when possible, in order to relate most of the aspects of the different formalisms. One of the reasons for selecting $\lambda\upsilon$ is that it is a calculus with a minimum number of rules and enjoying a good number of properties.

We are interested in *perpetual rewrite strategies*. A perpetual rewrite strategy is one that preserves the possibility of infinite derivations whenever possible. The interest in these strategies is that if they normalize a term M (compute a term N such that M β-rewrites to N and no further β-rewrites from N are possible) then this term is strongly normalizing, that is, *all* derivations starting from M are finite. In this work we define a perpetual rewrite strategy for $\lambda\upsilon$. Also, we use this perpetual strategy to prove that there is an inductive characterization of a class of terms in $\lambda\upsilon$ which captures exactly those that are SN in this calculus. By *inductive* we mean describing the set as the smallest set closed under some set of rules, as when defining the set of λ-terms or the set of theorems of some logic system (3).

Perpetuality will be proved using the *closure-tracing* and *minimal derivations* techniques. After that, and as an application, we give a set of deterministic inference rules which inductively characterize the set of SN $\lambda\upsilon$-terms and substitutions. Then, we use the preceding characterization to build an effective (decidable) perpetual reduction strategy for $\lambda\upsilon$ terms and substitutions.

A perpetual strategy allows to identify a subsystem of a calculus that restricts itself to some derivations which fulfill a specific condition, namely that reductions never delete a sub-term which is not SN.

Perpetuality in a named explicit substitution calculus has been first studied in (17).

4.1.1 Some remarks

The compatibility rules of $\lambda\upsilon$ play the role of the following classical lemma for the pure λ-calculus, used by van Raamsdonk et al. to prove perpetuality for the λ-calculus ((82), Lemma 3.1).

Lemma 4.1.1 *Let M, M', N, N' be λ-terms and let x be any variable. If $M \twoheadrightarrow_\beta M'$ and $N \twoheadrightarrow_\beta N'$ then $M\{x := N\} \twoheadrightarrow_\beta M'\{x := N'\}$.*

Proof: See (11). □

What is proved as a lemma in λ-calculus will hold in $\lambda\upsilon$ by compatibility of reduction.

Let $S(a)$ denote the set of sub-terms of term a for any TRS. We will use the following easy general result, which holds where indicated.

Remark 4.1.2 *In every Abstract Rewriting System (ARS) the first item holds, and in every TRS the following two items hold:*

1. *if $a \in SN$ and $a \to b$ then $b \in SN$*
2. *if $b \in SN$ and $a \in S(b)$ then $a \in SN$*

In chapter 1 we just gave the basic definitions about $\lambda\upsilon$, including term contexts in this calculus, which will be of use[1]. Proofs of PSN and confluence, as well as other results and comments, can be found in (13).

4.2 Perpetuality

In this section we will state and prove a version of the Perpetuality Lemma for $\lambda\upsilon$ using closure tracing along with minimal derivations techniques.

4.2.1 Auxiliary results

Now we prepare for proving perpetuality for $\lambda\upsilon$.

Definition 4.2.1 (set of sub-terms and sub-substitutions of a term/substitution) *Given a term a (resp., a substitution s), the set of sub-terms and sub-substitutions of a (resp., of s) is defined in an expected way:*

$$\begin{array}{lcl} S(m) & = & \{m\} \\ S(\lambda a) & = & \{\lambda a\} \cup S(a) \\ S(ab) & = & \{ab\} \cup S(a) \cup S(b) \\ S(a/) & = & \{a/\} \cup S(a) \\ S(\uparrow) & = & \{\uparrow\} \\ S(\Uparrow(s)) & = & \{\Uparrow(s)\} \cup S(s) \\ S(a[s]) & = & \{a[s]\} \cup S(a) \cup S(s) \end{array}$$

For every term a, $S(a)$ will include terms and substitutions (unless a is pure in which case it will consists of only terms).

Remark 4.2.2 *If $a \to_\upsilon b$, then there exists a sub-term of a of the form $c[s]$. This is due to the rules of $\lambda\upsilon$.*

We will use the following notion of skeleton of a term (introduced in (75) and also used in (17)).

[1] As we shall see, there will be no need to use substitution contexts in the present chapter.

Definition 4.2.3 (skeleton)

$$\begin{aligned} SK(m) &= m \\ SK(\lambda a) &= \lambda SK(a) \\ SK(ab) &= SK(a)SK(b) \\ SK(a[s]) &= SK(a)[\Box] \end{aligned}$$

The following definition was introduced in (13).

Definition 4.2.4 (internal and external reduction for $\lambda\upsilon$) *We define the internal reduction for $\lambda\upsilon$, denoted with $\rightarrow^{int}_{\lambda\upsilon}$, as the least sub-relation of $\rightarrow_{\lambda\upsilon}$ over terms and substitutions, which satisfies:*

1. $s \rightarrow_{\lambda\upsilon} s' \Rightarrow a[s] \rightarrow^{int}_{\lambda\upsilon} a[s']$
2. $a \rightarrow^{int}_{\lambda\upsilon} a' \Rightarrow ab \rightarrow^{int}_{\lambda\upsilon} a'b \wedge \ ba \rightarrow^{int}_{\lambda\upsilon} ba' \wedge \lambda a \rightarrow^{int}_{\lambda\upsilon} \lambda a'$

If $a \rightarrow_{\lambda\upsilon} b$ holds but $a \rightarrow^{int}_{\lambda\upsilon} b$ does not, we write $a \rightarrow^{ext}_{\lambda\upsilon} b$ instead, and call it an external reduction.

Remark 4.2.5 *$\rightarrow^{int}_{\lambda\upsilon}$ is a compatible reduction relation over Λ^t and Λ^s, but $\rightarrow^{ext}_{\lambda\upsilon}$ is not compatible.*

Definition 4.2.6 *(number of closures in a term or a skeleton)*

Given a term or substitution a we define the number of closures of a, noted $\#a$, in the following expected way:

$$\begin{aligned} \#m &= 0 \\ \#(ab) &= \#a + \#b \\ \#(\lambda a) &= \#a \\ \#a[s] &= 1 + \#a + \#s \\ \#(a/) &= \#a \\ \#(\uparrow) &= 0 \\ \#(\Uparrow(s)) &= \#s \end{aligned}$$

We straightforwardly extend it for skeletons, by defining $\#(\Box) = 0$.

Before the main lemmas, we give a very simple result that we will use afterwards.

Lemma 4.2.7 (Preservation of skeleton) *If $a \rightarrow^{int}_{\lambda\upsilon} b$ then $SK(a) = SK(b)$.*

PROOF: By induction on the inference of $a \rightarrow^{int}_{\lambda\upsilon} b$.

- If $a[s] \rightarrow^{int}_{\lambda\upsilon} a[s']$ with $s \rightarrow_{\lambda\upsilon} s'$, then $SK(a[s]) = SK(a)[\Box] = SK(a[s'])$.

- If $ab \rightarrow^{int}_{\lambda\upsilon} a'b$ with $a \rightarrow^{int}_{\lambda\upsilon} a'$, then $SK(ab) = SK(a)SK(b) =^{IH} SK(a')SK(b) = SK(a'b)$.
- If $ab \rightarrow^{int}_{\lambda\upsilon} ab'$ with $b \rightarrow^{int}_{\lambda\upsilon} b'$, then $SK(ab) = SK(a)SK(b) =^{IH} SK(a)SK(b') = SK(ab')$.
- If $\lambda a \rightarrow^{int}_{\lambda\upsilon} \lambda a'$ with $a \rightarrow^{int}_{\lambda\upsilon} a'$, then $SK(\lambda a) = \lambda SK(a) =^{IH} \lambda SK(a') = SK(\lambda a')$.

□

We will need to handle positions of a term and reductions over positions in a standard way as it was done in (13) and (17). Positions are elements of the set $\{1, 2\}^*$. Given a term or substitution a, we define the term at position p of a as usual (13). Let $Pos(a)$ be the set of all positions of term a. If the term at position p in a is the υ-redex used in a reduction $a \rightarrow_\upsilon a'$, we write $a \rightarrow^p_\upsilon a'$.

Recall we denote with $C\{\Box\}_p$ a context C in which the hole is located at the position p, and with $C\{a\}_p$ the context C where we replace the hole, which is located at the position p, by the term a.

We borrow the following three lemmas from (13). For the proofs, see this reference.

Lemma 4.2.8 (many-step Closure Tracing for $\lambda\upsilon$) *Let a_1, .., a_n ,$e \in \Lambda_\upsilon$ such that $a_i \rightarrow_{\lambda\upsilon} a_{i+1}$ for $1 \leq i \leq n-1$, and $a_n = C\{b[\Uparrow^j (e/)]\}_p$.*

Then:

1. *either $\exists$ i such that $a_i = C'\{(\lambda d')e'\}_{p'}$, with $e' \twoheadrightarrow_{\lambda\upsilon} e$*
2. *or $a_1 = C'\{d'[\Uparrow^k (e'/)]\}_{p'}$, with $e' \twoheadrightarrow_{\lambda\upsilon} e$.*

Lemma 4.2.9 (Projection Lemma for $\lambda\upsilon$) *The following hold:*

1. *$\forall a, b \in \Lambda_\upsilon$, if $a \rightarrow_{\lambda\upsilon} b$ then $\upsilon(a) \twoheadrightarrow_\beta \upsilon(b)$.*
2. *$\forall a, b \in \Lambda_\upsilon$, if $a \rightarrow^{ext}_{Beta} b$ then $\upsilon(a) \rightarrow_\beta \upsilon(b)$.*

Lemma 4.2.10 (Iterative Commutation Lemma for $\lambda\upsilon$) *Let a_0, .., $a_n \in \Lambda_\upsilon$ such that $\upsilon(a_0) \in SN_\beta$ and for $1 \leq i \leq n$ $\upsilon(a_0) = \upsilon(a_i)$ and*
$a_{i-1} \rightarrow^{int}_{\lambda\upsilon} * \,.\, \rightarrow^{ext}_\upsilon a_i$. Then $a_0 \rightarrow^{ext\ n}_\upsilon \,.\, (\rightarrow^{int}_{\lambda\upsilon} \,.\, \rightarrow^{ext}_\upsilon) * a_n$.

Lemma 4.2.10 states that some external υ-step could be brought to the beginning of such a derivation, but in the form of n external υ-steps, possibly followed by other steps which may be either internal $\lambda\upsilon$-steps or external υ-steps.

Lemma 13 in (13) states the following: if $a \in \Lambda_\upsilon$ is a pure term such that $a \in SN_\beta$, then for all infinite derivations $a = a_1 \rightarrow_{\lambda\upsilon} a_2 \rightarrow_{\lambda\upsilon} a_3 \rightarrow_{\lambda\upsilon} \ldots$ $\exists$n ≥ 1 such that $\forall$m $\geq$ n, $a_m \rightarrow^{int}_{\lambda\upsilon} a_{m+1}$. We prove now a generalization.

Lemma 4.2.11 *If $a \in \Lambda_\upsilon$ such that $\upsilon(a) \in SN_\beta$, then for all infinite derivations $a = a_1 \to_{\lambda\upsilon} a_2 \to_{\lambda\upsilon} a_3 \to_{\lambda\upsilon} \ldots$ $\exists n \geq 1$ such that $\forall m \geq n$, $a_m \to^{int}_{\lambda\upsilon} a_{m+1}$*

PROOF: Suppose there is an infinite derivation starting from a such that there are infinite external reductions.

If there is an infinite number of external (Beta)-reductions, such a derivation will have the form $a_1 \twoheadrightarrow_{\lambda\upsilon} a'_1 \to^{ext}_{Beta} a_2 \twoheadrightarrow_{\lambda\upsilon} a'_2 \to^{ext}_{Beta} a_3 \twoheadrightarrow_{\lambda\upsilon} \ldots$

By the Projection Lemma (1) and (2) we translate this derivations into

$\upsilon(a_1) \twoheadrightarrow_\beta \upsilon(a'_1) \to_\beta \upsilon(a_2) \twoheadrightarrow_\beta \upsilon(a'_2) \to_\beta \upsilon(a_3) \ldots$ so there is an infinite β-reduction starting from $\upsilon(a_1)$, but $\upsilon(a_1) \in SN_\beta$, which is an absurd. Therefore, all (Beta)-rewrites from one point onward are internal. So we can suppose that from one point onward, say a_m, there are no more external (Beta)-reductions. We will see that not only the (Beta)-rewrites, but also the υ-rewrites must be internal, by the following. Since υ is SN there exists a p such that no more than p υ- rewrites are possible from a_m. If we suppose that there are infinitely many external υ-rewrites in an infinite $\lambda\upsilon$ derivation starting from a_m, then there are of course at least $p+1$ of them. By the Iterative Commutation Lemma, (at least) $p+1$ external rewrites can be created starting from a_m, which is absurd since p was the maximum. □

Note that the above result is equivalent to stating that there exists a point such that all reductions from there on are internal (since a reduction cannot be both internal and external at the same time).

Also note that Lemma 13 in (13) does not directly imply our Lemma 4.2.11. This could be the case if for example one knows that there exists a pure term a' such that $a' \twoheadrightarrow_{\lambda\upsilon} a$ (and also a' SN). But this will not always happen (it is the problem of expansion to pure terms which will be treated in chapter 8).

Lemma 4.2.12 *Let $a \in \Lambda_\upsilon$ such that $\upsilon(a) \in SN_\beta$. Then for all infinite derivations $a = a_1 \to_{\lambda\upsilon} a_2 \to_{\lambda\upsilon} a_3 \to_{\lambda\upsilon} \ldots$ $\exists\ k \geq 1$, $p \in Pos(a)$, a context C, a term b and substitutions s_1, s_2, s_3, ... such that $s_1 \to_{\lambda\upsilon} s_2 \to_{\lambda\upsilon} s_3 \to_{\lambda\upsilon} \ldots$ and*
$a_1 \to_{\lambda\upsilon} a_2 \to_{\lambda\upsilon} \ldots \to_{\lambda\upsilon} a_k = C\{b[s_1]\}_p \to^{int}_{\lambda\upsilon} C\{b[s_2]\}_p \to^{int}_{\lambda\upsilon} C\{b[s_3]\}_p \to^{int}_{\lambda\upsilon} \ldots$

PROOF: Since $\upsilon(a) \in SN_\beta$, by the previous lemma $\exists k \geq 1$ such that $\forall i \geq k$, $a_i \to^{int}_{\lambda\upsilon} a_{i+1}$. By Lemma 4.2.7, $SK(a_i) = SK(a_{i+1}) = \ldots$

Since for all terms b we have that $\#b < \infty$, $\#\mathrm{SK}(a_i) < \infty$, and hence by König's Lemma there is a closure $b[s_1] \in \mathrm{S}(a_i)$ such that there is an infinite branch of rewrites in s_1. Then, $\exists$ k ≥ 1, p $\in$ Pos(a), a context C, a term b and substitutions s_1, s_2, s_3, ... such that
$a_1 \to_{\lambda\upsilon} a_2 \to_{\lambda\upsilon} \ldots \to_{\lambda\upsilon} a_k = \mathrm{C}\{b[s_1]\}_p \to^{int}_{\lambda\upsilon} \mathrm{C}\{b[s_2]\}_p \to^{int}_{\lambda\upsilon} \mathrm{C}\{b[s_3]\}_p \to^{int}_{\lambda\upsilon} \ldots$ □

Definition 4.2.13 (garbage) *In the reduction $a_1 \to_v a_2$ we call garbage to the erased (sub-)term or (sub-)substitution, if any. That is:*

1. *for (RVar) $(m+1)[a/] \to m$, its garbage is the term a*
2. *for (FVarLift) $1[\Uparrow(s)] \to 1$, its garbage is the substitution s*
3. *for the rules (Beta), (App), (Lam), (FVar), (RVarLift) and (VarShift), there is no garbage.*

Note that the garbage plays the same role as the eliminated sub-term by the (Gc)-rule in $\lambda\mathtt{x}$.

Definition 4.2.14 (safe reduction) *A reduction of any of the following forms will be called safe:*

$$\begin{array}{lcll}
(ab)[s] & \to & a[s]b[s] & \\
(\lambda a)[s] & \to & \lambda(a[\Uparrow(s)]) & \\
1[a/] & \to & a & \\
(m+1)[a/] & \to & m & if\ a \in SN_{\lambda v} \\
1[\Uparrow(s)] & \to & 1 & if\ s \in SN_{\lambda v} \\
(m+1)[\Uparrow(s)] & \to & m[s][\uparrow] & \\
m[\uparrow] & \to & m+1 &
\end{array}$$

We will use the symbol $\to_g$ to denote this relation extended to satisfy the usual compatibility. Note that $\to_g \subseteq \to_v$ is a proper inclusion.

Remark that safe reduction is equivalent to the fact that every garbage is SN.

The following Lemma states that for any v-reduction from a term of the form

$C\{b[\Uparrow^i(e/)]\}$, if the redex is not in e then that sub-term is either a sub-term of the reduct or a sub-term of the garbage.

Lemma 4.2.15 *Let C be a context, b, e terms and $i \geq 0$, such that $C\{b[\Uparrow^i(e/)]\} \to_v d$ where the reduction does not occur inside e. Then*

1. *either $e \in S(d)$*
2. *or the reduction garbage h is defined, with $e \in S(h)$*

PROOF: We use induction on the context C.

1. If C $= \Box$, we have two subcases:
   (a) If the redex is inside b, with $b \to_v b'$ then $d = b'[\Uparrow^i(e/)]$ therefore $e \in S(d)$.
   (b) Otherwise the redex must be $b[\Uparrow^i(e/)]$ (i.e., its position equal to the position of the hole in C) because the reduction does not occur inside e. Then we analyze each rule:

i. (App) $(uv)[\Uparrow^i (e/)] \to u[\Uparrow^i (e/)]v[\Uparrow^i (e/)]$, hence (1) applies.

ii. (Lam) $(\lambda a)[\Uparrow^i (e/)] \to \lambda(a[\Uparrow^{i+1} (e/)])$, hence (1) applies.

iii. (FVar) $1[\Uparrow^i (e/)] \to e$ with $i = 0$, hence (1) applies.

iv. (RVar) $(m+1)[\Uparrow^i (e/)] \to m$ with $i = 0$, hence (2) applies with $h = e/$.

v. (FVarLift) $1[\Uparrow^i (e/)] \to 1$ with $i \geq 1$, hence (2) applies with $h = \Uparrow^i (e/)$.

vi. (RVarLift) $(m+1)[\Uparrow^i (e/)] \to m[\Uparrow^{i-1} (e/)][\uparrow]$ with $i \geq 1$, hence (1) applies.

vii. (VarShift) $m[\uparrow] \to m+1$, but in this case the result holds vacuously since the redex does not have the required form $b[\Uparrow^i (e/)]$.

2. If $\mathrm{C} \neq \Box$, then we have the following cases:

(a) $C = C'v$, then the reduction step $C\{b[\Uparrow^i (e/)]\} \to_v d$ must be internal and $d = d_1 d_2$. There are two possibilities:

- $C'\{b[\Uparrow^i (e/)]\} \to_v d_1$ and $v = d_2$, hence the result follows by IH.
- $d_1 = C\{b[\Uparrow^i (e/)]\}$ and $v \to_v d_2$, hence (1) holds.

(b) $C = uC'$, analogous to the previous case.

(c) $C = \lambda C'$, then the reduction $C\{b[\Uparrow^i (e/)]\} \to_v d$ must be internal and the result follows by IH.

(d) $C = C'\{\Box\}[\Uparrow^k (\uparrow)]$ with $k \geq 0$, then we have the following cases:

- the reduction takes place at the root, then
  i. it cannot be (FVar), (RVar), (FVarLift), (RVarLift), nor (VarShift) since no index matches $C'\{b[\Uparrow^i (e/)]\}$.
  ii. if it is an (App)-step, clearly $e \in S(d)$ thus (1) holds.
  iii. if it is a (Lam)-step, clearly $e \in S(d)$ thus (1) holds.
- the reduction is internal in $C'\{b[\Uparrow^i (e/)]\}$, i.e. $C'\{b[\Uparrow^i (e/)]\}[\Uparrow^k (\uparrow)]$ $\to_v d'[\Uparrow^k (\uparrow)] = d$ with $C'\{b[\Uparrow^i (e/)]\} \to_v d'$, then since it cannot be inside e, by IH either $e \in S(d') \subseteq S(d)$, thus (1) holds, or $e \in S(h)$ thus (2) holds.

(e) $C = C'\{\Box\}[\Uparrow^k (f/)]$ with $k \geq 0$, then we have the following cases:

- the reduction takes place at the root, then we make the same considerations as before:
  i. it cannot be (FVar), (RVar), (FVarLift), (RVarLift), nor (VarShift) since again no index matches $C'\{b[\Uparrow^i (e/)]\}$.
  ii. if it is an (App)-step, clearly $e \in S(d)$ thus (1) holds.
  iii. if it is a (Lam)-step, clearly $e \in S(d)$ thus (1) holds.
- the reduction is inside $C'\{b[\Uparrow^i (e/)]\}$, i.e. $C'\{b[\Uparrow^i (e/)]\}[\Uparrow^k (f/)]$ $\to_v d'[\Uparrow^k (f/)] = d$ with $C'\{b[\Uparrow^i (e/)]\} \to_v d'$, then since it cannot be inside e, by IH either $e \in S(d') \subseteq S(d)$, thus (1) holds, or $e \in S(h)$ thus (2) holds.

- the reduction is internal in f, i.e. $C'\{b[\Uparrow^i (e/)]\}[\Uparrow^k (f/)]$ $\to_\upsilon C'\{b[\Uparrow^i (e/)]\}[\Uparrow^k (f'/)] = d$ with $f \to_\upsilon f'$, then clearly $e \in S(d)$ thus (1) holds.

(f) $C = u[\Uparrow^k (C'\{\Box\}/)]$ with $k \geq 0$, then we have the following cases:

- the reduction takes place at the root, then
  i. if it the rule applied is (FVar) ($k = 0$), then $e \in S(d)$ thus (1) holds.
  ii. if it is (RVar) ($k = 0$), then $e \in S(h)$ thus (2) holds.
  iii. if it is (FVarLift) ($k \geq 1$), then $e \in S(h)$ thus (2) holds.
  iv. if it is (RVarLift) ($k \geq 1$), then $e \in S(d)$ thus (1) holds.
  v. it cannot be (VarShift) since $\Uparrow^k (C'\{b[\Uparrow^i (e/)]\}/)$ does not match $\uparrow$.
  vi. if it is an (App)-step, clearly $e \in S(d)$ thus (1) holds.
  vii. if it is a (Lam)-step, clearly $e \in S(d)$ thus (1) holds.
- the reduction takes place at u, i.e. $u[\Uparrow^k (C'\{b[\Uparrow^i (e/)]\}/)]$ $\to_\upsilon u'[\Uparrow^k (C'\{b[\Uparrow^i (e/)]\}/)] = d$ with $u \to_\upsilon u'$, then clearly $e \in S(d)$ thus (1) holds.
- the reduction takes place inside the closure, i.e. at $C'\{b[\Uparrow^i (e/)]\}$, then $u[\Uparrow^k (C'\{b[\Uparrow^i (e/)]\}/)] \to_\upsilon u[\Uparrow^k (d'/)] = d$ with $C'\{b[\Uparrow^i (e/)]\} \to_\upsilon d'$, hence by IH $e \in S(d') \subseteq S(d)$ hence (1) holds, or $e \in S(h)$ hence (2) holds.

□

We will need to use minimal reductions in the way as it was done in (13) and (17).

Definition 4.2.16 (lexicographic order between reductions and minimal reductions) *Given an infinite derivation*

D: $a_1 \to_\upsilon^{p_1} a_2 \to_\upsilon^{p_2} \to \ldots$

and an infinite derivation

D': $b_1 \to_\upsilon^{q_1} b_2 \to_\upsilon^{q_2} \to \ldots$

we write D ∠ D' iff the sequence $(p_1, p_2, \ldots)$ *is lexicographically below the sequence* $(q_1, q_2, \ldots)$.

A minimal derivation is a derivation with no derivation below it (i.e., minimal for the strict total ordering ∠).

The following lemma holds in the context of (13). The only difference here is that a is any term and not necessarily a pure term.

Lemma 4.2.17 *If the set of infinite derivations from a given term a is nonempty (i.e.,* $a \in \infty$*) then there always exists a minimal infinite derivation.*

PROOF: Let $a_1 \to_{\lambda\upsilon} a_2 \to_{\lambda\upsilon} \dots$ be an infinite derivation. For every a_i, define the following subset of a_i positions: $P_i = \{p \in Pos(a_i) \mid$ if $a_i \to^p_{\lambda\upsilon} b$, then $b \in \infty\}$. Clearly P_i is nonempty and finite. For $i \geq 1$, let $p_i = min(P_i)$ any minimal element of the set of positions P_i with respect to the prefix ordering[1]. Then $a_1 \to^{p_1}_{\lambda\upsilon} a_2 \to^{p_2}_{\lambda\upsilon} \dots$ is a minimal infinite derivation. □

I.e., in each step, we select a position such that an infinite derivation "is still possible" and such that it is a minimal of such positions. Remark that a minimal infinite derivation always will exist among infinite derivations, but it may not be necessarily lesser than some finite derivation.

The following is one of the key lemmas in this section.

Lemma 4.2.18 *Suppose $a \to^p_\upsilon b$, and $b \in SN_{\lambda\upsilon}$, with the additional supposition that, if the garbage h is defined for this reduction, then $h \in SN_{\lambda\upsilon}$. Therefore $a \in SN_{\lambda\upsilon}$.*

PROOF: Let n = $\#a$. Note that since $a \to_\upsilon b$, by Remark 4.2.2 a has at least a closure, so $n \geq 1$.

We suppose there exists an infinite derivation from a, and apply induction on n.

1. Case n = 1.

   Let D := $a = a_1 \to_{\lambda\upsilon} a_2 \to_{\lambda\upsilon} \dots$ be a minimal infinite reduction from a (it always exists since the set of reductions is well-founded w.r.t. the lexicographic ordering). Since $b \in SN_{\lambda\upsilon}$ then $\upsilon(b) \in SN_\beta$, then as $a \to_\upsilon b$, $\upsilon(a) = \upsilon(b) \in SN_\beta$. Then we use Lemma 4.2.12 applied to a, and we have that $\exists$ k ≥ 1, p $\in$ Pos(a), a context C, a term d and substitutions s_1, s_2, s_3, ... such that

   $a = a_1 \to_{\lambda\upsilon} a_2 \to_{\lambda\upsilon} \dots \to_{\lambda\upsilon} a_k = \text{C}\{d[s_1]\}_p \to^{int}_{\lambda\upsilon} \text{C}\{d[s_2]\}_p \to^{int}_{\lambda\upsilon} \text{C}\{d[s_3]\}_p \to^{int}_{\lambda\upsilon} \dots$

   with $s_1 \to_{\lambda\upsilon} \dots \to_{\lambda\upsilon} s_k \to_{\lambda\upsilon} \dots$ Thus $\forall$i ≥ 1 $s_i \neq \Uparrow^k (\uparrow)$ (since $\forall$k $\Uparrow^k (\uparrow)$ has no redexes), so $\forall$i ≥ 1 $s_i = \Uparrow^k (e_i/)$ with $e_i \in \Lambda_\upsilon$, and therefore $e_1 \to_{\lambda\upsilon} e_2 \to_{\lambda\upsilon} \dots$, so $e_1 \in \infty_{\lambda\upsilon}$.

   By the many-step Closure Tracing Lemma, there are two possibilities:

   (a) Either $\exists$ j $\leq$ k, $p_j \in$ Pos(a), a context C', a term d' such that a_j = C'$\{(\lambda d')\ e_j\text{'}\ \}_{pj}$, with e_j' $\to^*_{\lambda\upsilon} e_j$. So we have another infinite derivation
   D' := $a_1 \to_{\lambda\upsilon} a_2 \to_{\lambda\upsilon} \dots \to_{\lambda\upsilon} a_j$ = C'$\{(\lambda d')\ e_j\text{'}\ \}_{pj}$
   $\to_{\lambda\upsilon}$ C'$\{d'[e_j\text{'}/]\ \}_{pj} \to^*_{\lambda\upsilon}$ C'$\{d'[e_j\ /]\}_{pj} \to_{\lambda\upsilon} \dots$
   satisfying D' $\angle$ D, thus contradicting the minimality of D.

   (b) Or $\exists r \geq 0$, d', c' such that $c' \twoheadrightarrow_{\lambda\upsilon} e_1$ and $d'[\Uparrow^r (c'/)] \in S(a_1)$ (i.e., the closure or some ancestor is already in a_1). Since $c' \twoheadrightarrow_{\lambda\upsilon} e_1 \in \infty_{\lambda\upsilon}$, $c' \in \infty_{\lambda\upsilon}$. We have two cases:

[1] It always exists since the sets $Pos(a_i)$ are finite. If there were more than one minimal element, it suffices to take the left-most between them.

i. $a = C'\{d'[\Uparrow^r (c'/)]\} \to_\upsilon b$, with the υ-redex different from $d'[\Uparrow^r (c'/)]$ (in d', in c' or elsewhere in C'). But then n would be greater than 1, contradicting the current assumption $n = 1$.

ii. The redex is $d'[\Uparrow^r (c'/)]$. Then, by Lemma 4.2.15, either $c' \in S(b)$, which implies that $c' \in SN_{\lambda\upsilon}$ by Remark 4.1.2(2) (since $b \in SN_{\lambda\upsilon}$); or $c' \in S(h)$ where h is the garbage of the reduction, but the additional requirement $h \in SN_{\lambda\upsilon}$ implies that $c' \in SN_{\lambda\upsilon}$ by Remark 4.1.2(2). In either case $c' \in SN_{\lambda\upsilon}$, contradicting the previous fact that $c' \in \infty_{\lambda\upsilon}$.

2. Case n > 1.

Let D := $a = a_1 \to_{\lambda\upsilon} a_2 \to_{\lambda\upsilon} \ldots$ be a minimal infinite reduction from a (it always exists since the set of reductions is well-founded). As before, since $b \in SN_{\lambda\upsilon}$ then $\upsilon(b) \in SN_\beta$, then as $a \to_\upsilon b$, $\upsilon(a) = \upsilon(b) \in SN_\beta$. Then we use Lemma 4.2.12 applied to a, and we have that $\exists$ k $\geq$ 1, p $\in$ Pos(a), a context C, a term d and substitutions s_1, s_2, s_3, ... such that

$a = a_1 \to_{\lambda\upsilon} a_2 \to_{\lambda\upsilon} \ldots \to_{\lambda\upsilon} a_k = \mathrm{C}\{d[s_1]\}_p \to^{int}_{\lambda\upsilon} \mathrm{C}\{d[s_2]\}_p \to^{int}_{\lambda\upsilon} \mathrm{C}\{d[s_3]\}_p \to^{int}_{\lambda\upsilon} \ldots$

with $s_1 \to_{\lambda\upsilon} s_2 \to_{\lambda\upsilon} \ldots \to_{\lambda\upsilon} s_k \to_{\lambda\upsilon} \ldots$ Thus $\forall$i $\geq$ 1 $s_i \neq\Uparrow^k(\uparrow)$ (since $\forall$k $\Uparrow^k (\uparrow)$ has no redexes), so $\forall$i $\geq$ 1 $s_i = \Uparrow^k (e_i/)$ with $e_i \in \Lambda_\upsilon$, and therefore $e_1 \to_{\lambda\upsilon} e_2 \to_{\lambda\upsilon} \ldots$, so $e_1 \in \infty_{\lambda\upsilon}$.

By the many-step Closure Tracing Lemma, there are two possibilities:

(a) Either $\exists$ j $\leq$ k, $p_j \in$ Pos(a), a context C', a term b such that $a_j = \mathrm{C'}\{(\lambda d')\ e_j\text{'}\ \}_{pj}$, with e_j' $\to^*_{\lambda\upsilon} e_j$. So we have another infinite derivation
D' := $a_1 \to_{\lambda\upsilon} a_2 \to_{\lambda\upsilon} \ldots \to_{\lambda\upsilon} a_j = \mathrm{C'}\{(\lambda d')\ e_j\text{'}\ \}_{pj}$
$\to_{\lambda\upsilon} \mathrm{C'}\{d'[e_j\text{'}/]\ \}_{pj} \to^*_{\lambda\upsilon} \mathrm{C'}\{d'[e_j\ /]\}_{pj} \to_{\lambda\upsilon} \ldots$
satisfying D' $\angle$ D, thus contradicting the minimality of D.

(b) Or $\exists$r, d', c' such that $c' \twoheadrightarrow_{\lambda\upsilon} e_1$ and $\Uparrow^r (d'[c'/]) \in S(a_1)$ (i.e., the closure or some ancestor is already in a_1). Since $c' \twoheadrightarrow_{\lambda\upsilon} e_1 \in \infty_{\lambda\upsilon}$, $c' \in \infty_{\lambda\upsilon}$. We have two cases:

i. the reduction step takes place in c', i.e., $a = C'\{d'[\Uparrow^r (c'/)]\} \to_\upsilon b$
$= C'\{d'[\Uparrow^r (c''/)]\}$ with $c' \to_\upsilon c''$. Since $b \in SN_{\lambda\upsilon}$, by Remark 4.1.2(2) $c'' \in SN_{\lambda\upsilon}$, and then by IH we have that $c' \in SN_{\lambda\upsilon}$, contradicting the previous fact that $c' \in \infty_{\lambda\upsilon}$.

ii. the reduction step does not take place in c'. Then, by Lemma 4.2.15, either $c' \in S(b)$, which implies that $c' \in SN_{\lambda\upsilon}$ by Remark 4.1.2(2) (since $b \in SN_{\lambda\upsilon}$); or $c' \in S(h)$ where h is the garbage of the reduction, but the additional requirement $h \in SN_{\lambda\upsilon}$ implies that $c' \in SN_{\lambda\upsilon}$ by Remark 4.1.2(2). In either case $c' \in SN_{\lambda\upsilon}$, contradicting the previous fact that $c' \in \infty_{\lambda\upsilon}$.

□

To the above perpetuality analysis we will add the case of Corollary 4.2.19(2) to allow the use of the (B) rule.

Note that the preceding proof fails if one considers the reduction of a term of the form $((\lambda a)b)[s_1]\ldots[s_k]a_1a_2\ldots a_n \to a[b/][s_1]\ldots[s_k]a_1a_2\ldots a_n$, since the (App) rule can be applied and then $((\lambda a')[s'_1]b'[s'_1])[s'_2]\ldots[s'_k]a'_1a'_2\ldots a'_n$ can appear in the derivation.

The previous perpetuality analysis has shown the following (recall Definition 4.2.14 for the $\to_g$ reduction).

Corollary 4.2.19 (Perpetuality Proposition for $\lambda\upsilon$) *The following hold:*

1. *Let c, $c' \in \Lambda_\upsilon$, with $c \to_g c'$. Then: $c \in SN_{\lambda\upsilon} \Leftrightarrow c' \in SN_{\lambda\upsilon}$.*
2. *For $n \geq 0$, $a[b/]a_1a_2\ldots a_n \in SN_{\lambda\upsilon} \Leftrightarrow (\lambda a)ba_1a_2\ldots a_n \in SN_{\lambda\upsilon}$*

PROOF:

1. ($\Rightarrow$) Trivially by Remark 4.1.2(1)

   ($\Leftarrow$) By Lemma 4.2.18.

2. ($\Leftarrow$) Trivially by Remark 4.1.2(1).

   ($\Rightarrow$) Suppose $a[b/]\ a_1a_2\ \ldots\ a_n \in SN_{\lambda\upsilon}$ and

   $(\lambda a)ba_1a_2\ \ldots\ a_n \notin SN_{\lambda\upsilon}$; thus, all a, b, a_1, a_2, $\ldots$, $a_n \in SN_{\lambda\upsilon}$ by Remark 4.1.2(2) and $(\lambda a)ba_1a_2\ \ldots\ a_n \in \infty$. By the $\lambda\upsilon$ rules any infinite derivation of $(\lambda a)ba_1a_2\ \ldots\ a_n$ must have the form

   $(\lambda a)ba_1a_2\ \ldots\ a_n \twoheadrightarrow (\lambda a')b'a'_1a'_2\ \ldots\ a'_n \to_{Beta} a'[b'/]a'_1a'_2\ \ldots\ a'_n \to \ldots$

   where $a \twoheadrightarrow a'$ and $b \twoheadrightarrow b'$, $\forall 1 \leq i \leq n\ a_i \twoheadrightarrow a'_i$ (for, since a, b, a_1, a_2, $\ldots$ $a_n \in SN_{\lambda\upsilon}$, none of them can have an infinite derivation).

   But then we also have that $a[b/]a_1a_2\ \ldots\ a_n \twoheadrightarrow a'[b'/]a'_1a'_2\ \ldots\ a'_n \to \ldots$

   which is an infinite derivation; so $a[b/]a_1a_2\ \ldots\ a_n \in \infty$, a contradiction since it was supposed to be SN.

□

4.2.2 Discussion

The meaning of the above results is that safe reductions preserves SN and can be closure-traced to SN terms. The reader should realize that the so defined safe reduction constitutes a subsystem of the original calculus when restricted to the set of terms admitting an infinite derivation. The condition of safe reduction cannot be loosened: the garbage should be SN,

otherwise the possibility of infinite reduction could be lost. The following term in $\lambda\upsilon$ is an example: $a = (\lambda 2)((\lambda 11)(\lambda 11))$, which is the equivalent of the classical $(\lambda x.y)\Omega$, where $\Omega = (\lambda x.xx)(\lambda x.xx)$. As it can be seen easily, $a \to 2[(\lambda 11)(\lambda 11)/] \to 1 \in SN_{\lambda\upsilon}$ but $2[(\lambda 11)(\lambda 11)/] \notin SN_{\lambda\upsilon}$.

Perpetuality will not be necessarily valid in other known explicit substitution calculi. Let us consider a calculus with composition, for example $\lambda\sigma$. Then, for instance, it will not be necessarily true that any infinite derivation of $(ab)[s_1][s_2]$ has the form $(ab)[s_1][s_2] \twoheadrightarrow (a'b')[s'_1][s'_2] \twoheadrightarrow (a'[s'_1]b'[s'_1])[s'_2] \twoheadrightarrow \ldots$ The reason is the existence of the composition rule which has the form $d[s][t] \to d[s \circ t]$, and therefore $(ab)[s_1][s_2] \to (ab)[s_1 \circ s_2] \to \ldots$ and we cannot ensure that a term of the form $(a'[s_1])(b'[s_1])[s'_2]$ will be eventually achieved in that derivation. Thus for $\lambda\sigma$, which is usually taken as the main representative of the family to where $\lambda\upsilon$ belongs, the present argument will not work.

We note also that we could have proceeded analogously defining the garbage always as a term (and not as a substitution), i.e., the garbage of $(m+1)[a/] \to m$ being the term a, for all $m \geq 1$, and the garbage of $1[\Uparrow^j (a/)] \to 1$ being the term a, for all $j \geq 1$. This will lead to the same result since, for every $j \geq 0$, $\Uparrow^j (a/) \in SN \Leftrightarrow a/ \in SN \Leftrightarrow a \in SN$.

4.3 A characterization of $\mathrm{SN}_{\lambda\upsilon}$

In this section we will treat a characterization of an interesting set of terms (actually a sub-ARS). We will pose formation rules for stating membership to that set.

Let us remark that a "formation rule" for a set is written as: the name of the rule, a list of preconditions, and a conclusion stating some formed term which should be a member of that set. Such a rule means that, if the premises are already in the set, and if the side condition (which will be indeed a syntactic condition on the premises and/or the conclusion) also holds, then the conclusion term will be in the set too. In some of the rules there might be no premises, or no side condition, or perhaps neither premises nor side condition. When we define a set using this style of rules, we mean that this set is the least set (with respect to set-theoretic inclusion) for which all the rules hold. Hence one application will be to prove properties of a set defined in such a fashion, which will be possible after this if we use induction, more precisely induction in the (length of the) proof that a given term is a member of this set.

Thus, in this section, as an application of the previous result, we formulate and prove for $\lambda\upsilon$ an inductive characterization of the $\mathrm{SN}_{\lambda\upsilon}$ set, analogous to the one presented in (82) for the pure λ calculus and the one in (17) for $\lambda\mathtt{x}$. The goal is to characterize inductively the family of SN terms for most calculi as to allow inductive proofs for these terms. This of course does not imply that such a set is recursive, but it gives a manner to prove that all its elements satisfy some given property.

4.3.1 Definition and remarks

For all $a \in SN_{\lambda\upsilon}$ we define *maxred(a)* to be the maximum length of any derivation from a in the usual way. In other words, maxred(a) = max{n $\in \mathbb{N}$ / $\exists a_1$, a_2, ..., $a_n \in \Lambda_\upsilon$ with $a \to a_1 \to a_2 \to \ldots \to a_n$}.

Note that maxred$(a) \geq 0$, and maxred$(a) = 0 \Leftrightarrow a \in \mathrm{NF}_{\lambda\upsilon}$. Note also that by the definition of Λ_υ we are defining maxred not only for terms but also for substitutions. We extend maxred to all Λ_υ defining it to be ∞ when applied to terms which are not in SN. In other words, maxred$(a) = \infty \Leftrightarrow a \in \infty_{\lambda\upsilon}$.

Definition 4.3.1 *Let $S_\upsilon \subseteq \Lambda_\upsilon$ be the smallest set closed under the following inference rules. In all of them, a, b, a_1, a_2, a_3, ..., a_n will denote terms of Λ_υ (i.e., elements of Λ_υ^t) and s_1, ..., s_k will denote substitutions (i.e., elements of Λ_υ^s). For every rule we assume $\forall n \geq 0$, $\forall k \geq 0$, $\forall m \geq 1$.*

1. $$\frac{a[b/]a_1a_2\ldots a_n \in S_\upsilon}{(\lambda a)ba_1a_2\ldots a_n \in S_\upsilon}\,\lambda - I$$

2. $$\frac{(a[s]b[s])[s_1]\ldots[s_k]a_1a_2\ldots a_n \in S_\upsilon}{ab[s][s_1]\ldots[s_k]a_1a_2\ldots a_n \in S_\upsilon}\,App$$

3. $$\frac{\lambda(a[\Uparrow(s)])[s_1]\ldots[s_k]a_1a_2\ldots a_n \in S_\upsilon}{(\lambda a)[s][s_1]\ldots[s_k]a_1a_2\ldots a_n \in S_\upsilon}\,\Uparrow -E$$

4. $$\frac{a[s_1]\ldots[s_k]a_1a_2\ldots a_n \in S_\upsilon}{1[a/][s_1]\ldots[s_k]a_1a_2\ldots a_n \in S_\upsilon}\,F[] - I$$

5. $$\frac{m[s_1]\ldots[s_k]a_1a_2\ldots a_n \in S_\upsilon, a \in S_\upsilon}{m{+}1[a/][s_1]\ldots[s_k]a_1a_2\ldots a_n \in S_\upsilon}\,R[] - I$$

6. $$\frac{1[s_1]\ldots[s_k]a_1a_2\ldots a_n \in S_\upsilon, s \in S_\upsilon}{1[\Uparrow(s)][s_1]\ldots[s_k]a_1a_2\ldots a_n \in S_\upsilon}\,F \Uparrow -I$$

7. $$\frac{m[s][\uparrow][s_1]\ldots[s_k]a_1a_2\ldots a_n \in S_\upsilon}{m{+}1[\Uparrow(s)][s_1]\ldots[s_k]a_1a_2\ldots a_n \in S_\upsilon}\,(R \Uparrow -I)\ or\ (\uparrow -E)$$

8. $$\frac{m{+}1[s_1]\ldots[s_k]a_1a_2\ldots a_n \in S_\upsilon}{m[\uparrow][s_1]\ldots[s_k]a_1a_2\ldots a_n \in S_\upsilon}\,\uparrow -I$$

9.
$$\frac{a \in S_v}{\lambda a \in S_v}\, Abs - I$$

10.
$$\frac{a \in S_v}{a/ \in S_v}\, / - I$$

11.
$$\frac{s \in S_v}{\Uparrow (s) \in S_v}\, \Uparrow - I$$

12.
$$\frac{}{\uparrow \in S_v}\, \uparrow - G$$

13.
$$\frac{a_1, a_2, \ldots, a_n \in S_v}{m a_1 a_2 \ldots a_n \in S_v}\, var - I$$

For convenience, we have adopted the rule names indicated at the right side of each one. In them, x-I stands for "x-introduction", x-E stands for "x-elimination", and x-G stands for "x-generation" (for rule $\uparrow$-G only).

S_v is nonempty, even without considering the rule (12), because the (var-I) rule serves as a startup rule since it is valid $\forall n \geq 0$. Thus, for instance, $m \in S_v\ \forall m \in \mathbb{N}$ (applying (var-I) for $n = 0$).

Let S be S_v. When from $c_1 \in S, c_2 \in S, \ldots, c_n \in S$ the r rule is applied to infer that $c \in S$, we will use the symbol $\vdash^r$ and write:

$c_1, c_2, \ldots, c_n \vdash^r c$.

Finally, when from $c_1 \in S, c_2 \in S, \ldots, c_n \in S$ some rule is applied to infer that $c \in S$, we will use the symbol $\vdash$ and write:

$c_1, c_2, \ldots, c_n \vdash c$.

4.3.2 Definitions and remarks

Remark 4.3.2 *For all $a \in S_v$ there exists $d \geq 0$ and some a_1,a_2, a_3, ..., a_d such that $a_d = a$ and $\forall 1 \leq i \leq d$,*

1. *either $a_i \in \mathbb{N}$ or $a_i = \uparrow$*
2. *or $\exists\ r_1, \ldots, r_n \in \mathbb{N}$, with $n \geq 0$, such that $\forall 1 \leq k \leq n$: $r_k < i$ and $a_{r1}, \ldots, a_{rn} \vdash a_i$*

Definition 4.3.3 *A sequence a_1,a_2, a_3, ..., a_d satisfying the above conditions, will be called an inference of $a \in S_v$, and we will write a_1,a_2, a_3, ..., $a_d \vdash a$. Its length will be d.*

Definition 4.3.4 *For $a \in S_v$ we define minder(a) = min{$d \in \mathbb{N}$ / there exists an inference of a with length d} = min{$d \in N$ / $\exists a_1$,a_2, a_3, ..., $a_d \in \Lambda_v$ such that a_1,a_2, a_3, ..., $a_d \vdash a$}*[1].

[1] We could extend the minder definition to all Λ_v, by defining minder(a)=∞ for $a \notin S_v$.

That is, minder(a) denotes the minimum length of any derivation of $a \in S_v$. A derivation of minimum length of a is called a *minimal derivation*. Note there always exist minimal derivations $\forall a \in S_v$ since the set of lengths of derivations of any $a \in S_v$ is a nonempty subset of $\mathbb{N}$, hence it has a minimum element. Note also that minder(a) is finite $\forall a \in S_v$.

4.3.3 Auxiliary results

Before proving the main result of this section, we will need the following easy auxiliary results.

Remark 4.3.5 *Let a, a_1,a_2, a_3, ..., a_n be terms, with $n \geq 0$, let s be a substitution and let $m \geq 1$. Then*

1. a_1,a_2, a_3, ..., $a_n \in SN_{\lambda v} \Leftrightarrow m a_1 a_2 \ldots \ a_n \in SN_{\lambda v}$
2. $a \in SN_{\lambda v} \Leftrightarrow \lambda a \in SN_{\lambda v}$
3. $a \in SN_{\lambda v} \Leftrightarrow a/ \in SN_{\lambda v}$
4. $s \in SN_{\lambda v} \Leftrightarrow \Uparrow (s) \in SN_{\lambda v}$

PROOF:

1. ($\Leftarrow$) Obvious by Remark 4.1.2(2) since all a_1,a_2, a_3, ..., a_n are sub-terms of $m a_1 a_2 \ldots \ a_n$.
   ($\Rightarrow$) In the infinite derivation $m a_1 a_2 \ldots \ a_n \rightarrow m a_1$'$a_2$' $\ldots \ a_n$' $\rightarrow \ldots$ the only redexes are inside the terms a_i, a_i', ...; therefore, by König's lemma, $\exists$ k, $1 \leq$ k $\leq$ n, such that $a_k \rightarrow a_k$' $\rightarrow \ldots$ is an infinite derivation, which is an absurd since $\forall 1 \leq$ i $\leq$ n, $a_i \in$ SN.
2. ($\Leftarrow$) Obvious by Remark 4.1.2(2) since a is a sub-term of λa.
   ($\Rightarrow$) In the infinite derivation $\lambda a \rightarrow \lambda a' \rightarrow \ldots$ the only redexes are inside the terms a, a', ...; then $a \rightarrow a' \rightarrow \ldots$ is an infinite derivation, which is an absurd since $a \in$ SN.
3. ($\Leftarrow$) Obvious by Remark 4.1.2(2) since a is a sub-term of $a/$.
   ($\Rightarrow$) In the infinite derivation $a/ \rightarrow a'/ \rightarrow \ldots$ the only redexes are inside the terms a, a', ...; then $a \rightarrow a' \rightarrow \ldots$ is an infinite derivation, which is an absurd since $a \in$ SN.
4. ($\Leftarrow$) Obvious by Remark 4.1.2(2) since s is a sub-term of $\Uparrow (s)$.
   ($\Rightarrow$) In the infinite derivation $\Uparrow (s) \rightarrow \Uparrow (s') \rightarrow \ldots$ the only redexes are inside the terms s, s', ...; then $s \rightarrow s' \rightarrow \ldots$ is an infinite derivation, which is an absurd since $s \in$ SN.

□

The idea behind the following lemma is that for every term one can "peel off" all applications leaving a term which can be an index, an abstraction or a closure, where in the last case we can again "peel off" all closures finally arriving to an index, an abstraction or an application.

Lemma 4.3.6 *Every term $c \in \Lambda_v^t$ has one and only one of the following forms:*

1. $c = ma_1a_2 \ldots a_n$, *with* $n \geq 0$
2. $c = (\lambda a)a_1a_2 \ldots a_n$, *with* $n \geq 0$
3. $c = a[s_1][s_2] \ldots [s_k]a_1a_2 \ldots a_n$ *with* $n \geq 0$, $k \geq 1$ *and* a *not a closure (i.e., a can be an index, an abstraction or an application)*

PROOF: By induction on c. □

Note we could replace the last clause by
3'. $c = a[s]\ a_1a_2 \ \ldots \ a_n$ with n ≥ 0 and a is any term
but we prefer the above formulation, which intuitively "peels off" first all applications and then all closures.

Let us justify the election of the rules. The following result will state that given a $SN_{\lambda v}$ term or substitution its inference tree can be deterministically constructed.

Lemma 4.3.7 *The above inference rules are deterministic, i.e., for all terms and substitutions $c \in S_v$ there exists a unique rule which could have been applied to obtain c. Therefore this is true for every step in any derivation of c.*

PROOF: Immediate consequence of Lemma 4.3.6. □

Lemma 4.3.8 *Let $c_1, \ldots, c_n$, $c \in SN_{\lambda v}$ with $n \geq 0$.*

1. *Let r be any rule among the rules (1) to (8). If $c_1 \vdash^r c$ then $maxred(c_1) < maxred(c)$*
2. *Let r be any rule among the rules (9) to (11). If $c_1 \vdash^r c$, then $maxred(c) = maxred(c_1)$.*
3. *Let r be any rule. If $c_1 \vdash^r c$ (resp., $c_1, c_2, \ldots, c_n \vdash^r c$), then $minder(c_1) < minder(c)$ (resp., $minder(c_i) < minder(c)$ for $1 \leq i \leq n$).*
4. *Consider the (var-I) rule. Then $\forall 1 \leq i \leq n$, $maxred(c_i) \leq maxred(c)$.*

PROOF:

1. In all cases we have that $c \to c_1$. Thus, if $c_1 \to a_1 \to a_2 \to a_3 \to \ldots \to a_d$ is a derivation of maximum length from c_1, then $c \to c_1 \to a_1 \to a_2 \to a_3 \to \ldots \to a_d$ is a derivation from c, therefore $maxred(c) \geq maxred(c_1) + 1 > maxred(c_1)$.

2. It trivially follows from the following facts that can be shown by easy induction in the length of the term a and in the length of the substitution s:

$$a \to a' \Leftrightarrow \lambda a \to \lambda a'$$
$$a \to a' \Leftrightarrow a/ \to a'/$$
$$s \to s' \Leftrightarrow \Uparrow(s) \to \Uparrow(s')$$

3. Using the fact that the rules are deterministic (Lemma 4.3.7).

4. Since any redex in $mc_1 \ldots c_n$ is a redex within some c_i.

□

Now we give the main result for $\mathrm{SN}_{\lambda v}$.

Proposition 4.3.9 *The above rules characterize $SN_{\lambda v}$, i.e., $S_v = SN_{\lambda v}$.*

PROOF: ($\subseteq$) Let $c \in S_v$. We will show that $c \in SN_{\lambda v}$ by induction in minder(c) (which is finite).

If minder(c) = 1, then either $c \in \mathbb{N}$ or $c = \uparrow$, and in either case $c \in SN_{\lambda v}$ trivially follows.

If minder(c) > 1, we have the following cases for c (one for each rule conclusion).

If c is a term:

1. $c = ma_1a_2 \ldots a_n$ where $a_1, a_2, \ldots, a_n \in S_v$, with n $\geq$ 0 and m $\geq$ 1; i.e., the (var-I) rule was last applied. So, since by Lemma 4.3.8 (4) $\forall 1 \leq$ i $\leq$ n minder(a_i) < minder(c), then by IH $a_1, a_2, \ldots, a_n \in SN_{\lambda v}$ and, by Remark 4.3.5 (1), $c \in SN_{\lambda v}$.
2. $c = \lambda a$ where $a \in S_v$; i.e., the (Abs-I) rule was last applied. So, since by Lemma 4.3.8 (3) minder(a) < minder(λa), then by IH, $a \in SN_{\lambda v}$ and, by Remark 4.3.5 (2), $\lambda a \in SN_{\lambda v}$.
3. $c = (\lambda a)ba_1a_2 \ldots a_n$ where $a[b/]a_1a_2 \ldots a_n \in S_v$ and $b \in S_v$, with n $\geq$ 0; i.e., the (λ-I) rule was last applied. So by IH, since by Lemma 4.3.8 (3) a minimal derivation of $a[b/]a_1a_2 \ldots a_n \in S_v$ has length less than the length of a minimal derivation of c, then $a[b/]a_1a_2 \ldots a_n \in SN_{\lambda v}$, and, since $c \to_g a[b/]a_1a_2 \ldots a_n$, then, by the Perpetuality Proposition (Corollary 4.2.19 (2)), $(\lambda a)ba_1a_2 \ldots a_n \in SN_{\lambda v}$.

For c having the form $a'[s]\,[s_1] \ldots [s_k]\,a_1a_2 \ldots a_n$ we can have:

1. $c = (\lambda a)\,[s]\,[s_1] \ldots [s_k]\,a_1a_2 \ldots a_n$ where $\lambda(a[\Uparrow(s)])\,[s_1] \ldots [s_k]\,a_1a_2 \ldots a_n \in S_v$ and $s \in S_v$, with k $\geq$ 0 and n $\geq$ 0; i.e., the ($\Uparrow$-E) rule was last applied. So by IH, since by Lemma 4.3.8 (3) a minimal derivation of $\lambda(a[\Uparrow(s)])\,[s_1] \ldots [s_k]\,a_1a_2 \ldots a_n \in S_v$ has length less than the length of a minimal derivation of c, then $\lambda(a[\Uparrow(s)])\,[s_1] \ldots [s_k]\,a_1a_2 \ldots a_n \in SN_{\lambda v}$, and, since $c \to_g \lambda(a[\Uparrow(s)])\,[s_1] \ldots [s_k]\,a_1a_2 \ldots a_n$, then by the Perpetuality Proposition (Corollary 4.2.19 (2)) $c \in \mathrm{SN}_{\lambda v}$.

2. c= (ab) $[s]$ $[s_1]$ ... $[s_k]$ a_1a_2 ... a_n where $(a[s]\ b[s])$ $[s_1]$ ... $[s_k] \in S_v$ and $s \in S_v$, with k ≥ 0 and n ≥ 0; i.e., the (App) rule was last applied. So by IH, since by Lemma 4.3.8 (3) a minimal derivation of $(a[s]\ b[s])$ $[s_1]$ ... $[s_k]$ a_1a_2 ... $a_n \in S$ has length less than the length of a minimal derivation of c, then $(a[s]\ b[s])$ $[s_1]$ ... $[s_k]$ a_1a_2 ... $a_n \in SN_{\lambda v}$, and, since $c \rightarrow_g (a[s]\ b[s])$ $[s_1]$ ... $[s_k]$ a_1a_2 ... a_n, then by perpetuality $c \in SN_{\lambda v}$.

3. c= 1 $[a\ /]$ $[s_1]$ ... $[s_k]$ a_1a_2 ... a_n where a $[s_1]$ ... $[s_k]$ a_1a_2 ... $a_n \in S_v$, with k ≥ 0 and n ≥ 0; i.e., the (F[]-I) rule was last applied. So by IH, since by Lemma 4.3.8 (3) a minimal derivation of a $[s_1]$ ... $[s_k]$ a_1a_2 ... $a_n \in S_v$ has length less than the length of a minimal derivation of c, then a $[s_1]$ ... $[s_k]$ a_1a_2 ... $a_n \in SN_{\lambda v}$, and, since $c \rightarrow_g a[s_1]$ ... $[s_k]$ a_1a_2 ... a_n, then by perpetuality $c \in SN_{\lambda v}$.

4. c= $(m+1)$ $[a/]$ $[s_1]$ ... $[s_k]$ a_1a_2 ... a_n where m $[s_1]$ ... $[s_k]$ a_1a_2 ... $a_n \in S_v$ and $a \in S_v$, with k ≥ 0, n ≥ 0 and m ≥ 1; i.e., the (R[]-I) rule was last applied. So by IH, since by Lemma 4.3.8 (3) a minimal derivation of m $[s_1]$ ... $[s_k]$ a_1a_2 ... $a_n \in S_v$ has length less than the length of a minimal derivation of c, then m $[s_1]$ ... $[s_k]$ a_1a_2 ... $a_n \in SN_{\lambda v}$, and, since minder(a) < minder(c), then $a \in SN_{\lambda v}$; then, since $c \rightarrow_g$ m $[s_1]$ ... $[s_k]$ a_1a_2 ... a_n, by perpetuality, $c \in SN_{\lambda v}$.

5. c= 1 $[\Uparrow (s)]$ $[s_1]$ ... $[s_k]$ a_1a_2 ... a_n where 1 $[s_1]$ ... $[s_k]$ a_1a_2 ... $a_n \in S_v$ and $s \in S_v$, with k ≥ 0 and n ≥ 0; i.e., the (F$\Uparrow$-I) rule was last applied. So by IH, since by Lemma 4.3.8 (3) a minimal derivation of 1 $[s_1]$ ... $[s_k]$ a_1a_2 ... $a_n \in S_v$ has length less than the length of a minimal derivation of c, then 1 $[s_1]$ ... $[s_k]$ a_1a_2 ... $a_n \in SN_{\lambda v}$, and, since minder(s) < minder(c), then $s \in SN_{\lambda v}$; then, since $c \rightarrow_g$ 1 $[s_1]$ ... $[s_k]$ a_1a_2 ... a_n, by perpetuality, $c \in SN_{\lambda v}$.

6. c= $(m+1)$ $[\Uparrow (s)]$ $[s_1]$ ... $[s_k]$ a_1a_2 ... a_n where m $[s]$ $[\uparrow]$ $[s_1]$ ... $[s_k]$ a_1a_2 ... $a_n \in S_v$ with k ≥ 0, n ≥ 0 and m ≥ 1; i.e., the (R$\Uparrow$-I) rule was last applied. So by IH, since by Lemma 4.3.8 (3) a minimal derivation of m $[s]$ $[\uparrow]$ $[s_1]$ ... $[s_k]$ a_1a_2 ... $a_n \in S_v$ has length less than the length of a minimal derivation of c, then m $[s]$ $[\uparrow]$ $[s_1]$ ... $[s_k]$ a_1a_2 ... $a_n \in SN_{\lambda v}$, and, since $c \rightarrow_g$ m $[s]$ $[\uparrow]$ $[s_1]$ ... $[s_k]$ a_1a_2 ... a_n, then by perpetuality $c \in SN_{\lambda v}$.

7. c= m $[\uparrow]$ $[s_1]$ ... $[s_k]$ a_1a_2 ... a_n where m + 1 $[s_1]$ ... $[s_k]$ a_1a_2 ... $a_n \in S_v$, with k ≥ 0, n ≥ 0 and m ≥ 1; i.e., the ($\uparrow$-I) rule was last applied. So, by IH, since by Lemma 4.3.8 (3) a minimal derivation of $(m+1)$ $[s_1]$ ... $[s_k]$ a_1a_2 ... $a_n \in S_v$ has length less than the length of a minimal derivation of c, then $(m+1)$ $[s_1]$ ... $[s_k]$ a_1a_2 ... $a_n \in SN_{\lambda v}$, and, since $c \rightarrow_g (m+1)$ $[s_1]$ ... $[s_k]$ a_1a_2 ... a_n, then by perpetuality $c \in SN_{\lambda v}$.

Otherwise, c is a substitution, and we have the following cases:

1. $c = a/$ where $a \in S_\upsilon$, i.e., the (/-I) rule was last applied. So, since by Lemma 4.3.8 (3) minder$(a) <$ minder(c) then, by IH, $a \in SN_{\lambda\upsilon}$, and, by Remark 4.3.5 (2), $c \in SN_{\lambda\upsilon}$.
2. $c =\Uparrow (s)$ where $s \in S_\upsilon$, i.e., the ($\Uparrow$-I) rule was last applied. So, since by Lemma 4.3.8 (3) minder$(s) <$ minder(c) then, by IH, $s \in SN_{\lambda\upsilon}$, and, by Remark 4.3.5 (4), $c \in SN_{\lambda\upsilon}$.
3. $c =\uparrow$, i.e., the $\uparrow$-G rule was applied in a minimal derivation. But $c \in SN_{\lambda\upsilon}$ since $\uparrow$ is a normal form.

($\supseteq$) Let $c \in SN_{\lambda\upsilon}$. We will show by lexicographic induction in (maxred(c), $|c|$) that $c \in S_\upsilon$.

If c is a term, by Lemma 4.3.6, we have the following possible cases:

1. $c = ma_1a_2 \ldots\ a_n$ with n ≥ 0 and m ≥ 1, and therefore by Remark 4.1.2(2), a_1, a_2, …, $a_n \in SN_{\lambda\upsilon}$. Because by Lemma 4.3.8 (4), $\forall 1 \leq$ i $\leq$ n (maxred$(a_i) \leq$ maxred(c) and $(maxred(a_i) = maxred(c) \Rightarrow |a_i| < |c|)$), then we can use the IH, so all a_1, a_2, …, $a_n \in S_\upsilon$; then, by the (var-I) rule, $ma_1a_2 \ldots\ a_n \in S_\upsilon$.
2. $c = \lambda a$, and therefore by Remark 4.1.2(2), $a \in$ SN. Since by Lemma 4.3.8(2) maxred$(c) =$ maxred(a), and $|a| < |a| + 1 = |c|$, then, by IH, $a \in S_\upsilon$; then, by the (Abs-I) rule, $c \in S_\upsilon$.
3. $c = (\lambda a)ba_1a_2 \ldots\ a_n$ with n ≥ 0, and therefore by Remark 4.1.2(2), a, b, $a_1a_2 \ldots\ a_n \in SN_{\lambda\upsilon}$. By Lemma 4.3.8(1), maxred$(a[b/]\ a_1a_2 \ldots\ a_n) <$ maxred(c), and then by IH $a[b/]\ a_1a_2 \ldots\ a_n \in S_\upsilon$;then, by the ($\lambda$-I) rule, $c \in S_\upsilon$.

For c having the form $a'[s]\ [s_1] \ldots\ [s_k]\ a_1a_2 \ldots\ a_n$ we can have:

1. $c = (\lambda a)\ [s]\ [s_1] \ldots\ [s_k]\ a_1a_2 \ldots\ a_n$ with k ≥ 0 and n ≥ 0, and therefore by Remark 4.1.2(2), s, s_1, s_2, …, s_k, a_1, a_2, …, $a_n \in SN_{\lambda\upsilon}$. By Lemma 4.3.8(1), maxred$(\lambda(a[\Uparrow (s)])\ [s_1] \ldots\ [s_k]\ a_1a_2 \ldots\ a_n) <$ maxred(c), and then by IH $\lambda(a[\Uparrow (s)])\ [s_1] \ldots\ [s_k]\ a_1a_2 \ldots\ a_n \in S_\upsilon$;then, by the ($\Uparrow$-E) rule, $c \in S_\upsilon$.
2. $c =$ (ab) $[s]\ [s_1] \ldots\ [s_k]\ a_1a_2 \ldots\ a_n$ with k ≥ 0 and n ≥ 0, and therefore by Remark 4.1.2(2), a, b, s, s_1, …, s_k, a_1, a_2, …, $a_n \in SN_{\lambda\upsilon}$. By IH, a, b, s, s_1, …, s_k, a_1, a_2, …, $a_n \in S_\upsilon$, and since by Lemma 4.3.8 (1) maxred $((a[s]\ b[s])\ [s_1] \ldots\ [s_k]\ a_1a_2 \ldots\ a_n) <$ maxred(c), $(a[s]\ b[s])\ [s_1] \ldots\ [s_k]\ a_1a_2 \ldots\ a_n \in S_\upsilon$; then, by the (App) rule, $c \in S_\upsilon$.
3. $c = 1[a\ /]\ [s_1] \ldots\ [s_k]\ a_1a_2 \ldots\ a_n$ with k ≥ 0 and n ≥ 0, and therefore by Remark 4.1.2(2), a, s_1, s_2, …, s_k, a_1, a_2, …, $a_n \in SN_{\lambda\upsilon}$. By Lemma 4.3.8 (1), maxred$(a[s_1] \ldots\ [s_k]\ a_1a_2 \ldots\ a_n) <$ maxred(c), and then by IH, since $a \in S_\upsilon$, by the (F[]-I) rule, $c \in S_\upsilon$.
4. $c = (m+1)\ [a/]\ [s_1] \ldots\ [s_k]\ a_1a_2 \ldots\ a_n$ with k ≥ 0, n ≥ 0 and m ≥ 1, and therefore by Remark 4.1.2(2), a, s_1, s_2, …, s_k, a_1, a_2, …, $a_n \in SN_{\lambda\upsilon}$. By Lemma 4.3.8 (1), maxred(m $[s_1] \ldots\ [s_k]\ a_1a_2 \ldots\ a_n) <$ maxred(c), and since maxred$(a) <$ maxred(c), then by IH $a \in S_\upsilon$, and m $[s_1] \ldots\ [s_k]\ a_1a_2 \ldots\ a_n \in S_\upsilon$; then, by the (R[]-I) rule, $c \in S_\upsilon$.

5. $c = 1\ [\Uparrow\ (s)]\ [s_1]\ \ldots\ [s_k]\ a_1a_2\ \ldots\ a_n$ with k ≥ 0 and n ≥ 0, and therefore by Remark 4.1.2(2), s, s_1, s_2, $\ldots$, s_k, a_1, a_2, $\ldots$, $a_n \in SN_{\lambda\upsilon}$. By Lemma 4.3.8 (1), maxred($1\ [s_1]\ \ldots\ [s_k]\ a_1a_2\ \ldots\ a_n$) $<$ maxred(c), and since maxred(s) $<$ maxred(c), then by IH $s \in S_\upsilon$, and $1\ [s_1]\ \ldots\ [s_k]\ a_1a_2\ \ldots\ a_n \in S_\upsilon$;then, by the (F$\Uparrow$-I) rule, $c \in S_\upsilon$.

6. $c = (m+1)\ [\Uparrow\ (s)]\ [s_1]\ \ldots\ [s_k]\ a_1a_2\ \ldots\ a_n$ with k ≥ 0, n ≥ 0 and m ≥ 1, and therefore by Remark 4.1.2(2), s, s_1, s_2, $\ldots$, s_k, a_1, a_2, $\ldots$, $a_n \in SN_{\lambda\upsilon}$. By Lemma 4.3.8 (1), maxred(m $[s]\ [\uparrow]\ [s_1]\ \ldots\ [s_k]\ a_1a_2\ \ldots\ a_n$) $<$ maxred(c), and then by IH m $[s]\ [\uparrow]\ [s_1]\ \ldots\ [s_k]\ a_1a_2\ \ldots\ a_n \in S_\upsilon$;then, by the (R$\Uparrow$-I) rule, $c \in S_\upsilon$.

7. $c =$ m $[\uparrow]\ [s_1]\ \ldots\ [s_k]\ a_1a_2\ \ldots\ a_n$ with k ≥ 0, n ≥ 0 and m ≥ 1, and therefore by Remark 4.1.2(2), s_1, s_2, $\ldots$, s_k, a_1, a_2, $\ldots$, $a_n \in SN_{\lambda\upsilon}$. By Lemma 4.3.8 (1), maxred($(m+1)\ [s_1]\ \ldots\ [s_k]\ a_1a_2\ \ldots\ a_n$) $<$ maxred(c), and then by IH $(m+1)\ [s_1]\ \ldots\ [s_k]\ a_1a_2\ \ldots\ a_n \in S_\upsilon$;then, by the ($\uparrow$-I) rule, $c \in S_\upsilon$.

Otherwise, c is a substitution, and we have the following cases:

1. $c = a/$, and therefore by Remark 4.1.2(2), $a \in SN_{\lambda\upsilon}$. Since by Lemma 4.3.8 (2), maxred(a) = maxred(c), and $|a| < |a| + 1 = |c|$, then, by IH $a \in S_\upsilon$;then, by the (/-I) rule, $c \in S_\upsilon$.
2. $c =\Uparrow\ (s)$, and therefore by Remark 4.1.2(2), $s \in SN_{\lambda\upsilon}$. Since by Lemma 4.3.8 (2), maxred(s) = maxred(c), and $|s| < |s| + 1 = |c|$, then, by IH $s \in S_\upsilon$;then, by the ($\Uparrow$-I) rule, $c \in S_\upsilon$.
3. $c =\uparrow$; then, by the ($\uparrow$-G) rule, $c \in S_\upsilon$.

□

4.3.4 Discussion

We have proved the previous inductive characterization of $\mathrm{SN}_{\lambda\upsilon}$ with the particularity that this set includes not only terms but also substitutions. This required a bigger number of cases to analyze but it was straightforward when grouping terms and substitutions in the same set.

As it can be seen, we have tried to keep the rule set as small as possible. We have some evidence that the given rule set cannot be substantially simplified or reduced. The rules are deterministic in a sense; this means that unique proofs of membership to $\mathrm{SN}_{\lambda\upsilon}$ can be easily obtained, although if the term is not in $\mathrm{SN}_{\lambda\upsilon}$, nothing can be done and the system will not provide any help since it is undecidable whether a given term is in the set of SN terms for all reasonable lambda calculi.

4.3.5 A perpetual reduction strategy for $\lambda\upsilon$

We recall (4; 17; 82) for the definition of *reduction strategies* and *perpetual reduction strategies* for TRSs, as well as discussions about their significance. For the sake of clarity we remind the following definition in the way we will use it.

Definition 4.3.10 *A perpetual reduction strategy for $\lambda\upsilon$ is a function $F : \Lambda_\upsilon \to \Lambda_\upsilon$ such that for all $a \in \Lambda_\upsilon$*

1. $a \in NF_\upsilon \Rightarrow F(a) = a$
2. $a \notin NF_\upsilon \Rightarrow a \to_{\lambda\upsilon} F(a)$
3. $a \in \Lambda_\infty \Rightarrow F(a) \in \Lambda_\infty$

When F satisfies only (1) and (2), it is just called a reduction strategy.

Note that, since this calculus is two-sorted, F is defined to be applied both to terms and substitutions, this being a difference with respect to $\lambda\mathtt{x}$.

We give here a perpetual reduction strategy for $\lambda\upsilon$ based in the previous characterization. We recall the notion of left-most redex and left-most reduction $\to_{\lambda\upsilon,l}$ (which we will not formalize in detail).

Definition 4.3.11 *We define $F : \Lambda_\upsilon \to \Lambda_\upsilon$ by the following:*

1. *If $u \in \Lambda_\upsilon - NF_\upsilon$, let $u = C\{\Delta\}$ where C is a context and $\Delta \in \Lambda_\upsilon$ is the left-most $\lambda\upsilon$-redex of u. In this case we define $F(u)$ as follows:*

   - *For terms:*

$$
\begin{array}{lcll}
F(C\{(\lambda a)b\\}) & = & C\{a[b/]\\} & \\
F(C\{(\lambda a)[s]\\}) & = & C\{\lambda a[\Uparrow (s)]\\} & \\
F(C\{(ab)[s]\\}) & = & C\{a[s]b[s]\\} & \\
F(C\{1[a/]\\}) & = & C\{a\\} & if\ a \in NF_\upsilon \\
F(C\{1[a/]\\}) & = & C\{1[F(a)/]\\} & if\ a \notin NF_\upsilon \\
F(C\{(m+1)[a/]\\}) & = & C\{m\\} & if\ a \in NF_\upsilon \\
F(C\{(m+1)[a/]\\}) & = & C\{(m+1)[F(a)/]\\} & if\ a \notin NF_\upsilon \\
F(C\{1[\Uparrow (s)]\\}) & = & C\{1\\} & if\ s \in NF_\upsilon \\
F(C\{1[\Uparrow (s)]\\}) & = & C\{1[\Uparrow (F(s))]\\} & if\ s \notin NF_\upsilon \\
F(C\{(m+1)[\Uparrow (s)]\\}) & = & C\{m[s][\uparrow]\\} & if\ s \in NF_\upsilon \\
F(C\{(m+1)[\Uparrow (s)]\\}) & = & C\{(m+1)[\Uparrow (F(s))]\\} & if\ s \notin NF_\upsilon \\
F(C\{m[\uparrow]\\}) & = & C\{m+1\\} &
\end{array}
$$

- *For substitutions:*

$$\begin{array}{lcl} F(\Uparrow(s)) & = & \Uparrow(F(s)) \\ F(b/) & = & F(b)/ \end{array}$$

2. *If $u \in NF_\upsilon$, we define $F(u) = u$. (For instance, $F(\uparrow) = \uparrow$.)*

Remark 4.3.12 *F is a reduction strategy (F satisfies (1) and (2) by simple inspection of each clause above.) Moreover, F is effective (i.e. decidable), since the computation of the left-most $\lambda\upsilon$-redex as well as the problem of whether a given term is a $\lambda\upsilon$-normal form are both decidable.*

Remark 4.3.13 *The strategy F defined above satisfies the following properties:*

1. *$F(ab) = F(a)b$ if the left-most redex of ab belongs to a*
2. *$F(ab) = a\ F(b)$ if the left-most redex of ab belongs to b*
3. *$F(\lambda a) = \lambda F(a)$*
4. *$F(a[s]) = F(a)[s]$ if the left-most redex of $a[s]$ belongs to a*
5. *$F(a[s]) = a[F(s)]$ if the left-most redex of $a[s]$ belongs to s*
6. *$F(\Uparrow(s)) = \Uparrow(F(s))$*

We will need the following lemma which states that F "follows" left-most redexes.

Lemma 4.3.14 *Let $a = C\{\Delta\} \in \Lambda_\upsilon$ - NF_υ where C is a context and $\Delta \in \Lambda_\upsilon$ is the left-most $\lambda\upsilon$-redex of a. Then $F(a) = C\{F(\Delta)\}$.*

PROOF: We use induction on C. We have the following cases:

1. C = $\Box$, then $a = \Box\{\Delta\} = \Delta$, then F($a$) = F($\Delta$) = C{F($\Delta$)} and we are done.
2. C = C'b, with C' a context. By IH F(C'{Δ}) = C'{F(Δ)}; then, using Remark 4.3.13 F(a) = F(C'{Δ}b) = F(C'{Δ})b = C'{F(Δ)} b = C{F(Δ)}
3. C = b C', with C' a context. By IH F(C'{Δ}) = C'{F(Δ)}; then, using Remark 4.3.13 F(a) = F(b C'{Δ}) = b F(C'{Δ}) = b C'{F(Δ)}= C{F(Δ)}
4. C = λC', with C' a context. By IH F(C'{Δ}) = C'{F(Δ)}; then, using Remark 4.3.13 F(a) = F(λ C'{Δ}) = λ F(C'{Δ}) = λ C'{F(Δ)}= C{F(Δ)}
5. C = C'[$\Uparrow^j$ ($\uparrow$)], with C' a context and $j \geq 0$. By IH F(C'{Δ}) = C'{F(Δ)}; then, using Remark 4.3.13 F(a) = F(C'{Δ} [$\Uparrow^j$ ($\uparrow$)]) = F(C'{Δ}) [$\Uparrow^j$ ($\uparrow$)] = C'{F(Δ)}[$\Uparrow^j$ ($\uparrow$)] = C{F(Δ)}

6. C = C'[⇑^{j} $(b/)$], with C' a context and $j \geq 0$. By IH F(C'{Δ}) = C'{F(Δ)}; then, using Remark 4.3.13 F(a) = F(C'{Δ} [⇑^{j} $(b/)$) = C'{F(Δ)}[⇑^{j} $(b/)$] = C{F(Δ)}
7. C = b [⇑^{j}(C' $/)$], with C' a context and $j \geq 0$. By IH F(C'{Δ}) = C'{F(Δ)}; then, iterating Remark 4.3.13 F(a) = F(b[⇑^{j}(C'{Δ} $/)$]) = b[⇑^{j}(C'{F(Δ)} $/)$] = C{F(Δ)}

□

We conclude the section with the following

Proposition 4.3.15 *The strategy F defined above is an effective perpetual reduction strategy for λv.*

Proof: As explained before, F is effective. Let us see that it is perpetual.

Let $c \in \Lambda_v$. Suppose F(c) $\in SN_{\lambda v}$, and let us prove that $c \in SN_{\lambda v}$. Since by Proposition 4.3.9 $S_v = \mathrm{SN}_{\lambda v}$, we reason by induction on the derivation of F(c) $\in S_v$ and show that $c \in S_v$.

If $c \in \mathrm{NF}_{\lambda v}$ then all is trivial. So we can suppose c has a left-most redex.

By Lemma 4.3.6 we have the following cases for c:

1. $c = ma_1a_2 \ldots\ a_n$ with n ≥ 0 and m ≥ 1. If n = 0, the result is trivial. Otherwise F(c) = $ma_1a_2 \ldots\ a_{i-1}$ F(a_i)$a_{i+1} \ldots\ a_n \in S_v$ (iterating Lemma 4.3.14), for some i s.t. a_1, ..., a_{i-1}, F(a_i), a_{i+1}, ..., $a_n \in S_v$. Then by IH $a_i \in S_v$. Then since all a_1, a_2, ..., $a_n \in S_v$, the (var-I) rules states that $c \in S_v$.
2. $c = \lambda a$. Then F(c) = λF(a) $\in S_v$ (using Lemma 4.3.14), where F(a) $\in S_v$, so by IH $a \in S_v$, and by the (Abs-I) rule, $c \in S_v$.
3. $c = (\lambda a)ba_1a_2 \ldots\ a_n$ with n ≥ 0. Then F(c) = (a[b/]) $a_1a_2 \ldots\ a_n \in S_v$, then by the ($\lambda$-I) rule, $c \in S_v$.
4. c = (ab) [s] [s_1] ... [s_k] $a_1a_2 \ldots\ a_n$ with n ≥ 0 and k ≥ 0. Then F(c) = (a[s] b[s]) [s_1] ... [s_k] $a_1a_2 \ldots\ a_n \in S_v$, then by the (App) rule, $c \in S_v$.
5. $c = (\lambda a)$ [s] [s_1] ... [s_k] $a_1a_2 \ldots\ a_n$ with n ≥ 0 and k ≥ 0. Then F(c) = λ(a[⇑ (s)]) [s_1] ... [s_k] $a_1a_2 \ldots\ a_n \in S_v$, then by the (⇑-E) rule, $c \in S_v$.
6. c = 1 [a /] [s_1] ... [s_k] $a_1a_2 \ldots\ a_n$ with n ≥ 0 and k ≥ 0, and $a \in \mathrm{NF}_{\lambda v}$. Then F($c$) = a[s_1] ... [s_k] $a_1a_2 \ldots\ a_n \in S_v$, then by the (F[]-I) rule, $c \in S_v$.
7. c = 1 [a /] [s_1] ... [s_k] $a_1a_2 \ldots\ a_n$ with n ≥ 0 and k ≥ 0, and $a \notin \mathrm{NF}_{\lambda v}$. Then F($c$) = 1 [F($a$)/][$s_1$] ... [$s_k$] $a_1a_2 \ldots\ a_n \in S_v$, then by IH and by the (F[]-I) rule, $c \in S_v$.
8. c = (m+1) [a/] [s_1] ... [s_k] $a_1a_2 \ldots\ a_n$ with n ≥ 0, k ≥ 0 and m ≥ 1, and $a \in \mathrm{NF}_{\lambda v}$. Then F($c$) = m [$s_1$] ... [$s_k$] $a_1a_2 \ldots\ a_n \in S_v$, then by the (R[]-I) rule, $c \in S_v$.

9. $c = (\mathrm{m}{+}1)\ [a/]\ [s_1]\ \ldots\ [s_k]\ a_1a_2\ \ldots\ a_n$ with $\mathrm{n} \geq 0$, $\mathrm{k} \geq 0$ and $\mathrm{m} \geq 1$, and $a \notin \mathrm{NF}_{\lambda\upsilon}$.
   Then $\mathrm{F}(c) = (\mathrm{m}{+}1)\ [\mathrm{F}(a)/]\ [s_1]\ \ldots\ [s_k]\ a_1a_2\ \ldots\ a_n \in S_\upsilon$, then by IH and by the (R[]-I) rule, $c \in S_\upsilon$.

10. $c = 1\ [\Uparrow (s)]\ [s_1]\ \ldots\ [s_k]\ a_1a_2\ \ldots\ a_n$ with $\mathrm{n} \geq 0$ and $\mathrm{k} \geq 0$, and $s \in \mathrm{NF}_{\lambda\upsilon}$.
   Then $\mathrm{F}(c) = 1\ [s_1]\ \ldots\ [s_k]\ a_1a_2\ \ldots\ a_n \in S_\upsilon$, then by the (F$\Uparrow$-I) rule, $c \in S_\upsilon$.

11. $c = 1\ [\Uparrow (s)]\ [s_1]\ \ldots\ [s_k]\ a_1a_2\ \ldots\ a_n$ with $\mathrm{n} \geq 0$ and $\mathrm{k} \geq 0$, and $s \notin \mathrm{NF}_{\lambda\upsilon}$.
   Then $\mathrm{F}(c) = 1\ [\Uparrow(\mathrm{F}(s))]\ [s_1]\ \ldots\ [s_k]\ a_1a_2\ \ldots\ a_n \in S_\upsilon$, then by IH and by the (F$\Uparrow$-I) rule, $c \in S_\upsilon$.

12. $c = (\mathrm{m}{+}1)\ [\Uparrow (s)]\ [s_1]\ \ldots\ [s_k]\ a_1a_2\ \ldots\ a_n$ with $\mathrm{n} \geq 0$, $\mathrm{k} \geq 0$ and $\mathrm{m} \geq 1$, and $s \in \mathrm{NF}_{\lambda\upsilon}$..
   Then $\mathrm{F}(c) = \mathrm{m}\ [s]\ [\uparrow]\ [s_1]\ \ldots\ [s_k]\ a_1a_2\ \ldots\ a_n \in S_\upsilon$, then by the (F$\Uparrow$-I) rule, $c \in S_\upsilon$.

13. $c = (\mathrm{m}{+}1)\ [\Uparrow (s)]\ [s_1]\ \ldots\ [s_k]\ a_1a_2\ \ldots\ a_n$ with $\mathrm{n} \geq 0$, $\mathrm{k} \geq 0$ and $\mathrm{m} \geq 1$, and $s \notin \mathrm{NF}_{\lambda\upsilon}$.
   Then $\mathrm{F}(c) = (\mathrm{m}{+}1)\ [\Uparrow(\mathrm{F}(s))]\ [s_1]\ \ldots\ [s_k]\ a_1a_2\ \ldots\ a_n \in S_\upsilon$, then by IH and by the (F$\Uparrow$-I) rule, $c \in S_\upsilon$.

14. $c = \mathrm{m}\ [\uparrow]\ [s_1]\ \ldots\ [s_k]\ a_1a_2\ \ldots\ a_n$ with $\mathrm{n} \geq 0$, $\mathrm{k} \geq 0$ and $\mathrm{m} \geq 1$.
   Then $\mathrm{F}(c) = (\mathrm{m}{+}1)\ [s_1]\ \ldots\ [s_k]\ a_1a_2\ \ldots\ a_n \in S_\upsilon$, then by the ($\uparrow$-I) rule, $c \in S_\upsilon$.

Otherwise, c is a substitution, and we have the following cases:

1. $c = a/$, then $\mathrm{F}(c) = \mathrm{F}(a)/ \in S_\upsilon$, where $\mathrm{F}(a) \in S_\upsilon$, then by IH $a \in S_\upsilon$, and by the (/-I) rule, $c \in S_\upsilon$.

2. $c = \Uparrow (s)$, then $\mathrm{F}(c) = \Uparrow(\mathrm{F}(s)) \in S_\upsilon$, where $\mathrm{F}(s) \in S_\upsilon$, then by IH $s \in S_\upsilon$, and by the ($\Uparrow$-I) rule, $c \in S_\upsilon$.

3. $c = \uparrow$, then by the ($\uparrow$-G) rule $c \in S_\upsilon$.

□

4.3.6 Digression

Note that for some cases the result could have been proved easily without the characterization, e.g., as in the following case

2) $c = \lambda a$

$\mathrm{F}(c) = \lambda\mathrm{F}(a) \in SN_{\lambda\upsilon}$, so by Remark 4.1.2(2) $\mathrm{F}(a) \in SN_{\lambda\upsilon}$, and by IH, $a \in SN_{\lambda\upsilon}$, then by Remark 4.3.5 (2), $c \in SN_{\lambda\upsilon}$.

And analogously for the substitution cases (1) and (2).

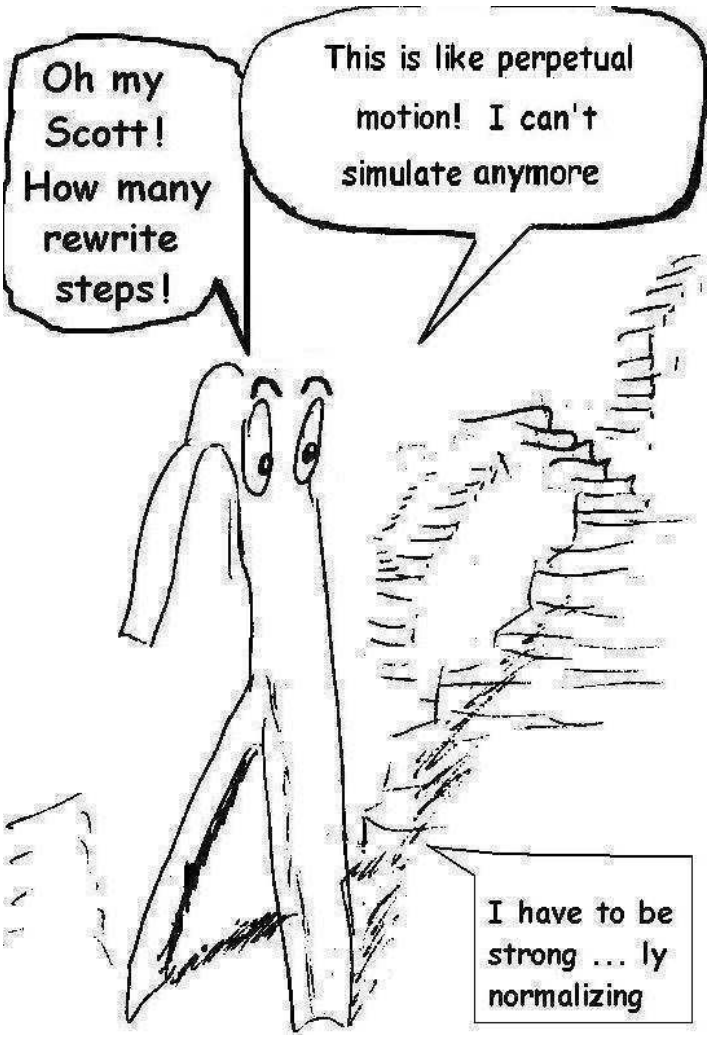
4.4 Conclusion and future work

We have formulated and proved perpetuality for λv. As an application we could state and prove a SN characterization and then give a perpetual reduction strategy. This has confirmed once more that such a reasonable de Bruijn explicit substitution calculus enjoys the expected properties when compared to a named calculus, and this leads to believe that other explicit substitution calculi (e.g., λs) preserve the same result. It is possible that most of these techniques could be applied to test a new calculus for this property in an analogous way. Another possible direction of work in the future could be to look for necessary and sufficient conditions under which various TRSs and calculi may enjoy perpetuality.

In the inference rules for characterizing SN terms, we have put together in the same set both terms and substitutions. This made the analysis simpler, although it could have been done separating both sorts and then formulating and proving the desired properties for the set of terms (and eventually for the set of substitutions). Also, in the characterization of SN terms, we had to decide which conditions to put in the rule premises. We also hope to find -or simplify- other characterizations of SN terms and substitutions, in several calculi, by analyzing different sets of rules, although we believe that these cannot be substantially changed.

We should remark that apart from the work in this chapter, a new method for finding perpetual reduction strategies was explored in a joint work (see (4)), namely the use of *zoom-in* strategies. This technique was applied to $\lambda\mathtt{x}$ and also to λw_s, a calculus with explicit substitution and weakening (see (26; 35)) in order to obtain perpetual strategies and rules characterizing their respective sets of SN terms.

Possible research includes comparing the reduction strategy F found above with others (eg., left-most outer-most, right-most inner-most, etc.) and to find normalizing as well as maximal reduction strategies (see (82)). Other tasks would be to find how to move from one perpetual strategy to another one, in order to access to different infinite derivations.


Oh my Scott! How many rewrite steps!
This is like perpetual motion! I can't simulate anymore
I have to be strong ... ly normalizing

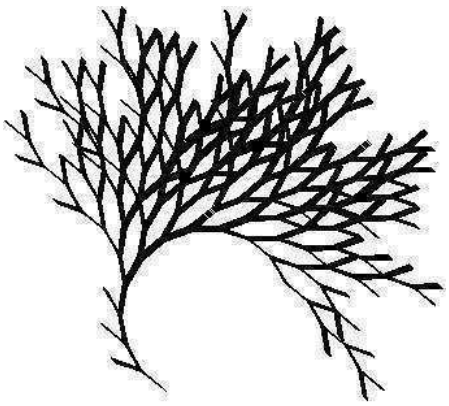


Figure 4.1: $(SS)(SS)(SS)(SS)(SS)(SS)(SS)(SS)$ after 28 left-most steps

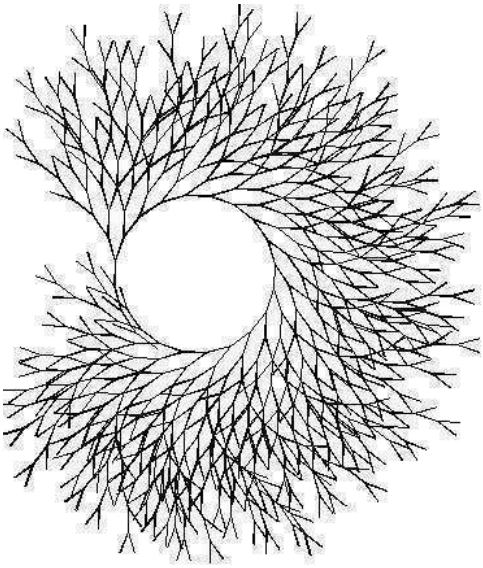


Figure 4.2: $(SS)(SS)(SS)(SS)(SS)(SS)(SS)(SS)$ after 65 left-most steps

Chapter 5

Extending lambda upsilon with composition

The method by which mathematics obtains its equations is the method of substitution. – L. Wittgenstein, *Tractatus Logico-Philosophicus*

ABSTRACT In this chapter a proposal for the $\lambda\upsilon$-calculus with composition rules (i.e. allowing the interaction of closures) is introduced and studied, and then weak confluence on open terms is proved. As an application, a derived variation of $\lambda\upsilon$ is presented with a unique de Bruijn index, preserving the properties of the previous extension.

5.1 Introduction

Since at present there is a huge number of calculi of explicit substitutions, with many variations and subtleties, we believe that a study must be done for comparing them from several points of view. This is strengthened by the fact that those calculi have properties which are different.

It is often useful to relate one calculus with another, by means of a mapping, that is an application preserving some of the calculus properties. Several mappings were proposed and used in the literature (15; 47; 75) with specific goals, for instance to let a calculus inherit a property which another one satisfies. But little effort has been done with respect to proving the non-existence of mappings between one and another. For instance, it could be interesting to find appropriate mappings from (and to) the $\lambda\sigma$-calculus (15; 66) i.e. to find out how $\lambda\upsilon$ (and also λs) substitutions (and maybe those of other calculi) could be represented in $\lambda\sigma$ which has a richer syntax.

One point when comparing different existing calculi is the expressive power of the substitution calculus (5), namely the characterization of the set of substitutions which they implement.

The reasons to study $\lambda\upsilon$ are multiple. It is a de Bruijn calculus, with minimal composition operators and rules. It has good properties, for instance the υ-calculus is canonical (CR and SN), and the calculus preserves strong normalization (PSN). It is two-sorted like $\lambda\sigma$, its syntax is simpler. It has a relatively small set of rules, hence some proofs can be generalized to those so called *basic substitution calculi* (BSC) (48) as well as CINNI (79) in which the constructs are of $\lambda\upsilon$-flavor.

When introducing rules propagating substitution over the different kinds of terms, composition rules may appear for closing critical pairs and gain (weak) confluence. The main goal of this chapter is to benefit from *composition-like* rules in the $\lambda\upsilon$-calculus (13; 60), in order to gain some good properties. Additionally, we explore the possibility of extending $\lambda\upsilon$ by other means.

Let us briefly discuss related work. One main development about confluence on open terms of an explicit substitution calculus was given in (45; 46) extending the λs-calculus. Some variants of $\lambda\upsilon$ have been previously given. In (31; 32), the calculi λd, λdn and λe are studied, and PSN is proved for the first time for a calculus with composition. These calculi are confluent on closed terms but not even weakly confluent on open terms, even when they handle rule schemas as we do. In (77), a calculus is given with one de Bruijn index and two "limited composition rules", which is also confluent on closed terms but not weakly confluent on open terms, since its rules are very limited.

Our work here consists mainly in two lines. One is to prove some negative results about $\lambda\upsilon$, which indeed are interesting to show that there are no appropriate mappings which relate it with some other calculi. The other one is to extend $\lambda\upsilon$ in order to gain good properties. The fact that $\lambda\upsilon$ lacks identity substitution motivates the addition of rules propagating closures, as will be done afterwards.

This chapter is presented as follows. Section 5.2 discusses some negative results about $\lambda\upsilon$ and introduces some straightforward extensions. Section 5.3 introduces $\lambda\upsilon_c$, an extension of the $\lambda\upsilon$-calculus which adds composition, resulting in a calculus with explicit substitution which is weakly confluent (WCR) on open terms. Section 5.4 shows a calculus with only one de Bruijn index having the same properties of $\lambda\upsilon_c$, based in the latter. Finally, we present our conclusion in section 5.5.

5.2 Some impossibilities of $\lambda\upsilon$.

We begin by showing some negative results about this calculus for justifying the extension. This is inspired in the idea of exploring which functions $\Lambda \to \Lambda$ (in particular, $\mathbb{N} \to \mathbb{N}$) are expressible in a calculus, and to be more precise, to see which kinds of structures they define.

One goal is to study the class of substitutions which can be represented in each of the calculi of explicit substitution.

We formulate the problem using notation for two-sorted calculi (in which closures make sense), although similar results can be achieved for calculi like λs.

Definition 5.2.1 *Given $\lambda\zeta$ a calculus of explicit substitution, where ζ is its associated substitution calculus, and given n substitutions $s_1, s_2, \ldots, s_m$, we will say that $f = f_{s_1,\ldots,s_m}$ is the function represented by the sequence of substitutions $s_1, \ldots, s_m$ if and only if*

1. *for every term a, $f(a)$ is in ζ-normal form*
2. *for every term a, $a[s_1]\ldots[s_m] \stackrel{*}{\rightarrow}_\zeta f(a)$*

Equivalently, $f(a) = \zeta(a[s_1]\ldots[s_m])$.

Definition 5.2.2 *We say that the substitution calculus ζ is closed under composition if, for every pair of functions f, g from terms to terms and for every pair of substitutions s and t such that f is represented by s and g is represented by t, there exists a substitution r such that $f\ o\ g$ is represented by r.*

One important property will be to find out if the substitution set is closed under composition. In all discussed calculi addition of a constant to an index is possible (by using the appropriate $\uparrow$ combination). We can say also that subtraction is available, in a limited way. Then one can ask if the same can be said about multiplication by a constant. For example, is there a substitution s such that $((1\ 2)3)[s] \stackrel{+}{\rightarrow} ((2\ 4)6)$, i.e. such that it multiplies by 2 all free indices? It would be interesting to discover if there is some relation with the expressive power of the calculus, its adequacy or its relation to λ-calculus. We shall see that the answer to the former is negative for known systems.

Our criterion when mapping the function meaning into the substitution apparatus is to consider normalization, eg. a function f from terms to terms in normal form will be expressible in the calculus whenever there is a substitution s such that for every term t the term $t[s]$ normalizes to $f(t)$.

We recall that the set of substitutions in the σ-calculus is closed under composition since there explicitly exists a composition operator, i.e. given substitutions s and s' (representing some functions f and g), $s\ o\ s'$ is another substitution in the calculus (representing the composition $f\ o\ g$). This happens to be false in other substitution calculi. Even when $a[s][s']$ (or a similar term) in most calculi computes the composition of s and s' applied to the term a, the substitution set needs not to be closed under composition.

The following is an interesting property for helping to find the functions represented by substitutions.

We will need the following lemma which is also proved in (60).

Lemma 5.2.3 $m[\Uparrow^i (s)] \twoheadrightarrow_\upsilon m$ *if* $m \leq i$, $i \geq 1$.

PROOF: By induction on m. If $m = 1$, it is straightforward since we have that $i \geq 1$ thus $1[\Uparrow^i (s)] \rightarrow_{FVarLift} 1$. If $m > 1$, we have that
$m[\Uparrow^i (s)] \rightarrow_{FVarLift} (m-1)[\Uparrow^{i-1} (s)][\uparrow]$
$\twoheadrightarrow (m-1)[\uparrow]$ (by IH, since $m - 1 \leq i - 1$)
$\rightarrow_{VarShift} m$. □

We will also need the following lemmas:

Lemma 5.2.4 $m[\Uparrow^i (\uparrow)] \twoheadrightarrow_\upsilon m + 1$ *if* $m > i \geq 0$.

PROOF: By induction on m. □

Lemma 5.2.5 $m[\Uparrow^i (b/)] \twoheadrightarrow_\upsilon m - 1$ *if* $m > i + 1$, $i \geq 1$.

PROOF: By induction on m. □

Proposition 5.2.6 *Let* ζ *be any of the following calculi:* υ, σ, *and let* $m \geq 1$. *Let* $f = f_{s_1,\ldots,s_m}$ *be the function represented by the sequence of substitutions* $s_1, \ldots, s_m$ *(cf. Definition 5.2.1). Then, there exist* $k \in \mathbb{N}, r \in \mathbb{Z}$ *depending on* $s_1, \ldots, s_m$ *such that for all* $n \geq k$ $f(n) = n + r$.

PROOF: We prove the assertion for υ although for σ something analogous can be done. Use induction on m, Remark 1.6.5 and Lemmas 5.2.4 and 5.2.5. Take $k = |s_1| + \cdots + |s_m| + 1$, and add up the different r's obtained, which results in an integer. □

Corollary 5.2.7 *Let* ζ *be any of the following calculi:* υ, σ, *and let* $m \geq 1$. *Let* $f = f_{s_1,\ldots,s_m}$ *be the function represented by the sequence of substitutions* $s_1, \ldots, s_m$. *Then,* $f(\mathbb{N})$ *is an infinite set.*

PROOF: Immediate from Proposition 5.2.6 □

As a consequence we have the following impossibilities.

Corollary 5.2.8 *Let* ζ *be any of the following calculi:* υ, σ *and let* $k, r, m \geq 1$. *Then, there do not exist*

1. *substitutions* $s_1, \ldots, s_m$ *such that for all* $n \in \mathbb{N}$, $n[s_1] \ldots [s_m] \xrightarrow{*}_\zeta k$.
2. *substitutions* $s_1, \ldots, s_m$ *such that for all* $n \in \mathbb{N}$, $n[s_1] \ldots [s_m] \xrightarrow{*}_\zeta k$ *if* n *is odd,* $n[s_1] \ldots [s_m] \xrightarrow{*}_\zeta r$ *if* n *is even.*
3. *substitutions* $s_1, \ldots, s_m$ *such that for all* $n \in \mathbb{N}$, $n[s_1] \ldots [s_m] \xrightarrow{*}_\zeta k$ *if* $n \leq m$, $n[s_1] \ldots [s_m] \xrightarrow{*}_\zeta r$ *if* $n > m$.

PROOF: All these are special cases of Corollary 5.2.7. □

Now we consider the *identity* function. The presence or absence of the identity function in the substitution set of a given explicit substitution calculus is a subtle question. For instance, $\lambda\sigma$-calculus has the identity as a given substitution in a "primitive" way. But that is not the case of $\lambda\upsilon$-calculus as we shall see.

We have

Proposition 5.2.9 *υ does not have an identity substitution (even restricting its application to indices), i.e. there does not exist s such that for all $n \in \mathbb{N}$, $n[s] \stackrel{+}{\rightarrow}_\upsilon n$.*

PROOF: Suppose there exists such a substitution s. Then we reason by cases and use the above lemmas:

- $s = \Uparrow^m (b/)$, then taking for instance $n = m + 2$ one has that $n[s] \stackrel{+}{\rightarrow}_\upsilon m + 1$ which is a υ-normal form, so $n[s]$ can never reduce to n.
- $s = \Uparrow^m (\uparrow)$, then taking for instance $n = m + 1$ one has that $n[s] \stackrel{+}{\rightarrow}_\upsilon m + 2$, and we reason as above.

Thus for all s there exists n such that $n[s]$ does not reduce to n. □

Let id denote the identity substitution in $\lambda\sigma$. For pure terms in $\lambda\sigma$ we have the useful and well-known

Lemma 5.2.10 *For all $a \in \Lambda\sigma^t$ pure, $a[id] \stackrel{*}{\rightarrow}_\sigma a$*

PROOF: By induction on a. □

As an application we show the non-existence of an appropriate mapping from $\lambda\sigma$ to $\lambda\upsilon$ in the following sense. Let us denote the set of $\lambda\sigma$ terms with $\Lambda\sigma^t$, and the set of $\lambda\sigma$ substitutions with $\Lambda\sigma^s$.

Proposition 5.2.11 *There does not exist a pair of functions (t, t') such that $t : \Lambda\sigma^t \rightarrow \Lambda^t_\upsilon$, with $\mathbb{N} \subseteq t(\Lambda\sigma^t)$ i.e. the image of t includes all de Bruijn indices, $t' : \Lambda\sigma^s \rightarrow \Lambda^s_\upsilon$, and such that*

1. *$a \rightarrow_\sigma b \Rightarrow t(a) \stackrel{*}{\rightarrow}_\upsilon t(b)$*
2. *$t(a[s]) = t(a)[t'(s)]$*

PROOF: Suppose there exists such a pair (t, t'), take $s = t'(id)$. Then, for all pure a, by (2) we would have $t(a[id]) = t(a)[s]$, but $a[id] \xrightarrow{*}_{\sigma} a$ by Lemma 5.2.10, then by (1) $t(a[id]) \xrightarrow{*}_{\upsilon} t(a)$ thus $t(a)[s] \xrightarrow{*}_{\upsilon} t(a)$. Since given any $m \in I\!N$ by hypothesis there exists $a \in \Lambda\sigma^t$ such that $m = t(a)$, then s would be an identity substitution in $\lambda\upsilon$, which is absurd by Proposition 5.2.9. □

For pure terms in $\lambda\mathbf{x}$ we have

Lemma 5.2.12 *For all $M \in \Lambda$, $M < x := x > \xrightarrow{*}_{x} M$*

PROOF: By induction on M. □

Then, as another application we show the non-existence of a mapping from $\lambda\mathbf{x}$ to $\lambda\upsilon$ in the following sense.

Proposition 5.2.13 *There does not exist a pair of functions (t, t') such that $t : \Lambda x \to \Lambda^t_{\upsilon}$ with $I\!N \subseteq t(\Lambda x)$ i.e. the image of t includes all de Bruijn indices, $t' : V \times \Lambda x \to \Lambda^s_{\upsilon}$, and such that*

1. *$M \to_x N \Rightarrow t(M) \xrightarrow{*}_{\upsilon} t(N)$*
2. *$t(M < x := N >) = t(M)[t'(x, N)]$*

PROOF: Suppose there exists such a pair (t, t'), take $s = t'(x, x)$, then, for all pure M and x, by (2) we would have $t(M < x := x >) = t(M)[s]$, but $M < x := x > \xrightarrow{*}_{x} M$ by Lemma 5.2.12, then by (1) $t(M < x := x >) \xrightarrow{*}_{\upsilon} t(M)$ thus $t(M)[s] \xrightarrow{*}_{\upsilon} t(M)$. Since given any $m \in I\!N$ by hypothesis there exists $M \in \Lambda x$ such that $m = t(M)$, then s would be an identity substitution in $\lambda\upsilon$, which is again absurd by Proposition 5.2.9. □

The previous results become useful since mappings between de Bruijn calculi usually need to map indices to indices (actually themselves). The non-existence of mappings may indicate that a simulation is not suitable for transferring results from one calculus to another one.

Proposition 5.2.13 is to be generalized in chapter 8. Namely, we will prove the non-existence of good mappings with less restrictions. To achieve this, further analysis will be done. We refine now the negative result of Proposition 5.2.9. υ does not have any generic identity substitution for the entire set of terms, as formalized next.

We will call a *generic substitution context* in $\lambda\upsilon$ any context of the form $C\{\Box\} = \Box[s_1] \ldots [s_n]$ where the hole should be replaced by a $\lambda\upsilon$ term. A *generic identity substitution* will be a generic substitution context such that for all $a \in \Lambda^t_{\upsilon}$, $C\{a\} \xrightarrow{+}_{\upsilon} a$

Lemma 5.2.14 *Let $a, b \in \Lambda^t_{\upsilon}$, $s, t \in \Lambda^s_{\upsilon}$.*

1. *If $\lambda a \to_{\upsilon} b$, then there exists $b' \in \Lambda^t_{\upsilon}$ such that $b = \lambda b'$ and $a \to_{\upsilon} b'$.*

2. *If* $\lambda a \to_v \lambda b$, *then* $a \to_v b$.

3. *If* $\Uparrow(s) \to_v t$, *then there exists* $t' \in \Lambda_v^s$ *such that* $t = \Uparrow(t')$ *and* $s \to_v t'$.

PROOF:

1. By induction on a, checking all v-rules and possible redex positions.
2. Using the previous item.
3. By induction on s, checking all v-rules and possible redex positions.

□

Lemma 5.2.15 *Let* $n \geq 1$, $s_1, \ldots, s_n \in \Lambda_v^s$ *such that* $a[s_1]\ldots[s_n] \xrightarrow{*}_v a$ *for all* $a \in \Lambda$ *(i.e. pure terms). Then* $a[\Uparrow(s_1)]\ldots[\Uparrow(s_n)] \xrightarrow{*}_v a$.

PROOF: Since $a[s_1]\ldots[s_n] \xrightarrow{*}_v a$ for all a, then in particular $(\lambda a)[s_1]\ldots[s_n] \xrightarrow{*}_v \lambda a$, but $(\lambda a)[s_1]\ldots[s_n] \xrightarrow{*}_v \lambda(a[\Uparrow(s_1)]\ldots[\Uparrow(s_n)])$, then by Lemma 5.2.14 and confluence of v (13; 60), $a[\Uparrow(s_1)]\ldots[\Uparrow(s_n)] \xrightarrow{*}_v a$ since a is pure. □

Lemma 5.2.16 *Let* $n \geq 1$, $t_1, \ldots, t_n \in \Lambda_v^s$.

1. *If* $1[1/][\Uparrow(t_1)]\ldots[\Uparrow(t_n)] \to_v c$, *then*
   - *either* $c = 1[\Uparrow(t_1)]\ldots[\Uparrow(t_n)]$,
   - *or there exists* $1 \leq i \leq n$ *and* $t_i' \in \Lambda_v^s$ *such that* $c = 1[1/][\Uparrow(t_1)]\ldots[\Uparrow(t_{i-1})][\Uparrow(t_i')][\Uparrow(t_{i+1})]\ldots[\Uparrow(t_n)]$ *with* $t_i \to_v t_i'$.
2. *If* $1[\Uparrow(t_1)]\ldots[\Uparrow(t_n)] \to_v c$, *then*
   - *either* $c = 1[\Uparrow(t_2)]\ldots[\Uparrow(t_n)]$,
   - *or there exists* $1 \leq i \leq n$ *and* $t_i' \in \Lambda_v^s$ *such that* $c = 1[\Uparrow(t_1)]\ldots[\Uparrow(t_{i-1})][\Uparrow(t_i')][\Uparrow(t_{i+1})]\ldots[\Uparrow(t_n)]$ *with* $t_i \to_v t_i'$.
3. *If* $1[1/][\Uparrow(t_1)]\ldots[\Uparrow(t_n)] \xrightarrow{*}_v c$, *then*
   - *either* $c = 1[1/][\Uparrow(t_1')]\ldots[\Uparrow(t_n')]$ *with* $t_i \xrightarrow{*}_v t_i'$ *for* $1 \leq i \leq n$,
   - *or* $c = 1[\Uparrow(t_k')]\ldots[\Uparrow(t_n')]$, *with* $t_i \xrightarrow{*}_v t_i'$ *for* $k \leq i \leq n$, *with* $1 \leq k$.
   - *or* $c = 1$.

PROOF: For items 1 and 2, checking all v-rules and possible redex positions. Item 3 follows from the iteration of the previous items. □

As a consequence, unless one considers pure terms only, there are no generic identities as stated next:

Proposition 5.2.17 *Given any $n \geq 1$, there do not exist $s_1, \ldots, s_n \in \Lambda_\upsilon^s$ such that for all $a \in \Lambda_\upsilon^t$ $a[s_1]\ldots[s_n] \xrightarrow{+}_\upsilon a$, i.e., υ does not have generic identity substitutions.*

PROOF: Suppose that there exist such substitutions. By Lemma 5.2.15 we may assume that all $s_i = \Uparrow (t_i)$ for some t_i. Take $a = 1[1/]$. Thus by Lemma 5.2.16 the set of reducts of $a[s_1]\ldots[s_n]$ is

- $1[1/][\Uparrow (t'_1)]\ldots[\Uparrow (t'_n)]$
- $1[\Uparrow (t'_1)]\ldots[\Uparrow (t'_n)]$
- $1[\Uparrow (t'_2)]\ldots[\Uparrow (t'_n)]$
- $\ldots$
- $1[\Uparrow (t'_{n-2})][\Uparrow (t'_{n-1})][\Uparrow (t'_n)]$
- $1[\Uparrow (t'_{n-1})][\Uparrow (t'_n)]$
- $1[\Uparrow (t'_n)]$
- 1

for all possible t'_i such that $t_i \xrightarrow{*}_\upsilon t'_i$, $1 \leq i \leq n$. But we also have that $1[1/][s_1]\ldots[s_n] \xrightarrow{*}_\upsilon 1[1/]$, and $1[1/]$ does not match any of the above terms. This is absurd. □

Although we omit details here, these negative assertions can be extended to CINNI which is of the $\lambda\upsilon$-style (cf. (79)).

Definition 5.2.18 *A* weak identity *is a context C such that $C\{a\} \xrightarrow{+} a$ for all $a \in \Lambda$.*

Weak identities in principle are not necessarily of the form $\Box[s_1]\ldots[s_n]$ i.e. an iteration of closures, but we are specially interested in them.

Although there are no generic identities, there are weak identities in υ as given by the

Lemma 5.2.19 *υ has an infinite number of weak identities*

PROOF: Let $C\{\Box\} = \Box[\Uparrow^k (\uparrow)][\Uparrow^k (b/)]$ for any fixed term $b \in \Lambda_\upsilon^t$ and $k \in \mathbb{N}$. Then by induction on a, $C\{a\} \xrightarrow{+}_\upsilon a$ for all $a \in \Lambda$. The case $a = n \in \mathbb{N}$ an index is proved by cases, separately analyzing $n \leq k$ and $n > k$ and using Lemmas 5.2.3, 5.2.4 and 5.2.5. □

Note that the above lemma does not contradict Proposition 5.2.17 since it applies to pure terms. This lemma will motivate an interaction rule in the next section. The following is a simple consequence of the above results:

Corollary 5.2.20 *The set of υ substitutions is not closed under composition.*

PROOF: In υ it is enough to recall that, even restricted to pure terms, the identity function is not expressible with a single substitution (Lemma 5.2.9), although it can be represented by a sequence of two closures (last lemma). □

Thus in the following section we tackle this problem partially, that is, we add composition-like rules to $\lambda\upsilon$. We say partially because not every pair of closures will interact, but the necessary ones to gain weak confluence on open terms.

To finish this subsection, we add as a curiosity that there is still another kind of weak identity whose existence is shown next.

Definition 5.2.21 *A* local identity *is a function p from terms to substitutions, $p : \Lambda^t_\upsilon \to \Lambda^s_\upsilon$, such that for all $a \in \Lambda^t_\upsilon$, $a[p(a)] \xrightarrow{+}_\upsilon a$. A* pure local identity *is a function p from pure terms to substitutions, $p : \Lambda \to \Lambda^s_\upsilon$, such that for all $a \in \Lambda$, $a[p(a)] \xrightarrow{+}_\upsilon a$.*

Lemma 5.2.22 *υ has an infinite number of pure local identities which are not weak identities.*

PROOF: For all $a \in \Lambda\upsilon$ we define $p_m(a) = \Uparrow^m (\uparrow)$ for $m \in \mathbb{N}$, and $l(a) \in \mathbb{N}$ inductively as follows:

$$\begin{array}{lcl} l(n) & = & n \\ l(\lambda a) & = & 1 + l(a) \\ l(ab) & = & max(l(a), l(b)) \\ l(a[s]) & = & max(l(a), l(s)) \\ l(\Uparrow (s)) & = & l(s) \\ l(\uparrow) & = & 1 \\ l(a/) & = & l(a) \end{array}$$

It can be shown by induction on a that $a[p_m(a)] \xrightarrow{+}_\upsilon a$ for all $m \geq l(a)$, again using the above lemmas. □

Note that in general it is not the case that $a[\Uparrow^m (\uparrow)] \xrightarrow{+}_\upsilon a$ if $m < l(a)$.

It is also interesting that it is impossible to extend this result for all $\Lambda\upsilon$ terms, shown below.

Lemma 5.2.23 *There does not exist $p : \Lambda^t_\upsilon \to \Lambda^s_\upsilon$ such that for all $a \in \Lambda\upsilon$, $a[p(a)] \xrightarrow{+}_\upsilon a$. I.e., there are no local identities in $\lambda\upsilon$.*

PROOF: We provide a counterexample: let $a = 1[\uparrow][\Uparrow ((\lambda 1)/)]$. [1] Now suppose there is such a p, so let $s = p(a)$, then it must be the case that $a[s] \xrightarrow{+}_\upsilon a$. Let us write all the $\xrightarrow{*}_\upsilon$-reducts of $a[s]$ (which are possibly infinite):
$1[\uparrow][\Uparrow ((\lambda 1)/)][s']$

[1] It seems that there is no simpler one!

$2[\Uparrow ((\lambda 1)/)][s']$
$1[(\lambda 1)/)][\uparrow][s']$
$(\lambda 1)[\uparrow][s]$
$(\lambda 1[\Uparrow (\uparrow)])[s']$
$(\lambda 1[\Uparrow (\uparrow)][\Uparrow (s')])$
$(\lambda 1)[s']$
$(\lambda 1[\Uparrow (s')])$
$\lambda 1$
for every substitution s' such that $s \xrightarrow{*}_{\upsilon} s'$.

But then, as a consequence of Lemma 5.2.14, a does not appear among these terms. Then $a[s] \xrightarrow{+}_{\upsilon} a$ does not hold, therefore there is no such a function p. □

So this kind of identity cannot be extended to all terms. As a morale, we have seen that in a substitution calculus (even if it is a BSC) the existence of weak identities and pure local identities do not imply the existence of identities nor of local identities.

5.3 Extending $\lambda\upsilon$ with composition

In most of the proofs of the previous section, the main problem and difficulty come from the fact that in $\lambda\upsilon$ closures do not interact, that is, a closure may block certain "desirable" reductions.

The objective of this section is to exhibit composition rules for $\lambda\upsilon$ and show that local confluence on open terms is obtained. The reasons for introducing a $\lambda\upsilon$ with composition are twofold. On one hand, we have seen in the previous section that the difficulty of getting an identity substitution comes from the non interaction of closures. The other one is to gain weak confluence as we pointed above.

We will see that the composition rules will not have side conditions (e.g. inequalities, like λs_e), nevertheless these interaction rules will be actually *rule schemas* instead of plain rules.

Definition 5.3.1 *We introduce the $\lambda\upsilon_c$-calculus (a variation of the $\lambda\upsilon$-calculus) on open terms given by the following syntax:*

$$\textbf{Open Terms } a \ ::= \ n \mid X \mid (aa) \mid (\lambda a) \mid a[s]$$

$$\textbf{Open Substitutions } s \ ::= \ x \mid a/ \mid \uparrow \mid \Uparrow (s)$$

where $n \geq 1$, X denotes term meta-variables and x denotes substitution meta-variables; and with the following rules (where the first eight constitute the $\lambda\upsilon$-calculus):

$$(\lambda a)b \to a[b/] \tag{Beta}$$

$$1[a/] \to a \tag{FVar}$$

$$(n+1)[a/] \to n \tag{RVar}$$

$$1[\Uparrow (s)] \to 1 \tag{FVarLift}$$

$$(n+1)[\Uparrow (s)] \to n[s][\uparrow] \tag{RVarLift}$$

$$n[\uparrow] \to n+1 \tag{VarShift}$$

$$(ab)[s] \to a[s]b[s] \tag{App}$$

$$(\lambda a)[s] \to \lambda(a[\Uparrow (s)]) \tag{Lam}$$

to which we add the following composition rules schemas:

$$a[\Uparrow^i (\uparrow)][\Uparrow^i (b/)] \to a \quad i \geq 0 \tag{c1}$$

$$a[\Uparrow^i (b/)][\Uparrow^i (s)] \to a[\Uparrow^{i+1} (s)][\Uparrow^i (b[s]/)] \quad i \geq 0 \tag{c2}$$

$$a[\Uparrow^i (\uparrow)][\Uparrow^{i+1} (s)] \to a[\Uparrow^i (s)][\Uparrow^i (\uparrow)] \quad i \geq 0 \tag{c3}$$

Let us denote the set of open terms with Λ^t_υ and the set of open substitutions with Λ^s_υ.

We define $\lambda\upsilon_c$ as $\lambda\upsilon$-rules plus {(c1), (c2), (c3)}. υ plus {(c1),(c2),(c3)} i.e. all $\lambda\upsilon_c$-rules except (Beta) is called υ_c.

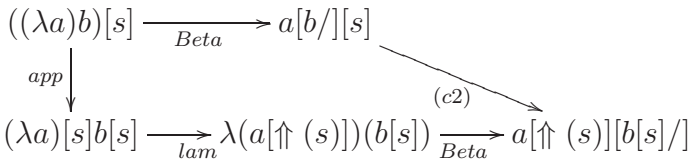
We added rules (c1), (c2) and (c3) in order to close the critical pairs. As we said these are rule schemas since they state reduction relations for a (denumerable) family of terms, anyway with these rules, given any term, its redexes can be easily calculated.

Using Lemmas 5.2.3,5.2.4 and 5.2.5 we have the following

Proposition 5.3.2 *$\lambda\upsilon_c$ is WCR on open terms.*

Proof: We check all the critical pairs created by the addition of the new rules.

- (Beta) with (App):



- (FVarLift) with (c1) where $i \geq 1$:

$$\begin{array}{ccc} 1[\Uparrow^i (\uparrow)][\Uparrow^i (b/)] & \xrightarrow{c1} & 1 \\ \downarrow & \nearrow & \\ 1[\Uparrow^i (b/)] & & \end{array}$$

- (RVarLift) with (c1) where $i \geq 1$:

$$\begin{array}{ccccccc} (n+1)[\Uparrow^i (\uparrow)][\Uparrow^i (b/)] & & & \xrightarrow[c1]{} & & & n+1 \\ \downarrow{\scriptstyle RVarLift} & & & & & & \| \\ n[\Uparrow^{i-1} (\uparrow)][\uparrow][\Uparrow^i (b/)] & \xrightarrow[Case]{} & n[\uparrow][\Uparrow^i (b/)] & \xrightarrow[VarShift]{} & (n+1)[\Uparrow^i (b/)] & \xrightarrow[L.5.2.3]{} & n+1 \\ \downarrow{\scriptstyle n \geq i, L.5.2.4} & & {\scriptstyle n \leq i-1, L.5.2.3} & & & & \| \\ (n+1)[\uparrow][\Uparrow^i (b/)] & \xrightarrow[VarShift]{} & (n+2)[\Uparrow^i (b/)] & & \xrightarrow[\text{L. } 5.2.5]{} & & n+1 \end{array}$$

- (VarShift) with (c1) ($i = 0$):

$$\begin{array}{ccc} n[\uparrow][b/] & \xrightarrow[c1]{} & n \\ \downarrow & & \| \\ (n+1)[b/] & \longrightarrow & n \end{array}$$

- (App) with (c1):

$$\begin{array}{ccc} (ab)[\Uparrow^i (\uparrow)][\Uparrow^i (c/)] & \xrightarrow[c1]{} & (ab) \\ \downarrow & & \uparrow \\ (a[\Uparrow^i (\uparrow)]b[\Uparrow^i (\uparrow)])[\Uparrow^i (c/)] & & \uparrow \\ \downarrow & & \uparrow \\ (a[\Uparrow^i (\uparrow)][\Uparrow^i (c/)])(b[\Uparrow^i (\uparrow)][\Uparrow^i (c/)]) & \twoheadrightarrow & a[\Uparrow^i (\uparrow)][\Uparrow^i (c/)]b[\Uparrow^i (\uparrow)][\Uparrow^i (c/)] \end{array}$$

- (Lam) with (c1):

$$\begin{array}{ccc} (\lambda a)[\Uparrow^i (\uparrow)][\Uparrow^i (b/)] & \xrightarrow[c1]{} & (\lambda a) \\ \downarrow & & \uparrow \\ (\lambda a[\Uparrow^{i+1} (\uparrow)])[\Uparrow^i (b/)] & \longrightarrow & \lambda(a[\Uparrow^{i+1} (\uparrow)][\Uparrow^{i+1} (b/)]) \end{array}$$

- (FVar) with (c2) ($i = 0$):

$$\begin{array}{ccc} 1[a/][s] & \xrightarrow[c2]{} & 1[\Uparrow(s)][a[s]/] \\ \downarrow & \swarrow & \\ a[s] & & \end{array}$$

- (RVar) with (c2) ($i = 0$):

$$\begin{array}{ccc} (n+1)[a/][s] & \xrightarrow[c2]{} & (n+1)[\Uparrow(s)][a[s]/] \\ {\scriptstyle RVar}\downarrow & & \downarrow \\ n[s] & \xleftarrow{c1} & n[s][\uparrow][a[s]/] \end{array}$$

- (FVarLift) with (c2) ($i \geq 1$):

$$\begin{array}{ccc} 1[\Uparrow^i(b/)][\Uparrow^i(s)] & \xrightarrow[c2]{} & 1[\Uparrow^{i+1}(s)][\Uparrow^i(b[s]/)] \\ {\scriptstyle FVarLift}\downarrow & & \downarrow{\scriptstyle FVarLift} \\ 1[\Uparrow^i(s)] & \longrightarrow & 1 \end{array}$$

- (RVarLift) with (c2) ($i \geq 1$):

$$\begin{array}{ccc} (n+1)[\Uparrow^i(b/)][\Uparrow^i(s)] & \xrightarrow[c2]{} & (n+1)[\Uparrow^{i+1}(s)][\Uparrow^i(b[s]/)] \\ {\scriptstyle RVarLift}\downarrow & & \downarrow{\scriptstyle RVarLift} \\ n[\Uparrow^{i-1}(b/)][\uparrow][\Uparrow^i(s)] = lhs & & n[\Uparrow^i(s)][\uparrow][\Uparrow^i(b[s]/)] = rhs \end{array}$$

  - Case $n = 1,\ i = 1$:

    $lhs \rightarrow b[\uparrow][\Uparrow(s)] \twoheadrightarrow b[s][\uparrow]$

    $rhs \twoheadrightarrow b[s][\uparrow]$

  - Case $n = 1,\ i > 1$:

$$lhs \longrightarrow 1[\uparrow][\Uparrow^i(s)] \longrightarrow 2[\Uparrow^i(s)] \longrightarrow 1[\Uparrow^{i-1}(s)][\uparrow] \xrightarrow{i>1} 2$$

$$rhs \twoheadrightarrow 2[\Uparrow^i(b[s]/)] \longrightarrow 1[\Uparrow^{i-1}(b[s]/)][\uparrow] \xrightarrow{i>1} 2$$

  - Case $n > 1,\ i = 1$:

$$\begin{array}{ccc} lhs = n[b/][\uparrow][\Uparrow(s)] & & rhs = n[\Uparrow(s)][\uparrow][\Uparrow(b[s]/)] \\ \downarrow & & \downarrow \\ n[b/][s][\uparrow] & \xrightarrow[c2]{} & n[\Uparrow(s)][b[s]/][\uparrow] \end{array}$$

  - Case $n > 1,\ i > 1$:

$$
\begin{array}{ccc}
lhs = n[\Uparrow^{i-1}(b/)][\uparrow][\Uparrow^{i}(s)] & & \\
\downarrow{\scriptstyle c3} & & \\
n[\Uparrow^{i-1}(b/)][\Uparrow^{i-1}(s)][\uparrow] & & rhs = n[\Uparrow^{i}(s)][\uparrow][\Uparrow^{i}(b[s]/)] \\
\downarrow{\scriptstyle c2} & \swarrow{\scriptstyle c3} & \\
n[\Uparrow^{i}(s)][\Uparrow^{i-1}(b[s]/)][\uparrow] & &
\end{array}
$$

- (App) with (c2):

$$
\begin{array}{ccc}
(ab)[\Uparrow^{i}(c/)][\Uparrow^{i}(s)] & \xrightarrow[c2]{} & (ab)[\Uparrow^{i+1}(s)][\Uparrow^{i}(c[s]/)] \\
\downarrow{\scriptstyle app} & & \downarrow{\scriptstyle app} \\
((a[\Uparrow^{i}(c/)])(b[\Uparrow^{i}(c/)]))[\Uparrow^{i}(s)] & & (a[\Uparrow^{i+1}(s)]b[\Uparrow^{i+1}(s)])[\Uparrow^{i}(c[s]/)] \\
\downarrow{\scriptstyle app} & & \downarrow{\scriptstyle app} \\
(a[\Uparrow^{i}(c/)][\Uparrow^{i}(s)])(b[\Uparrow^{i}(c/)][\Uparrow^{i}(s)]) & \xrightarrow[c2]{} & (a[\Uparrow^{i+1}(s)][\Uparrow^{i}(c[s]/)])(b[\Uparrow^{i+1}(s)][\Uparrow^{i}(c[s]/)])
\end{array}
$$

- (Lam) with (c2):

$$
\begin{array}{ccc}
(\lambda a)[\Uparrow^{i}(c/)][\Uparrow^{i}(s)] & \xrightarrow[c2]{} & (\lambda a)[\Uparrow^{i+1}(s)][\Uparrow^{i}(c[s]/)] \\
\downarrow{\scriptstyle lam} & & \downarrow{\scriptstyle lam} \\
(\lambda a[\Uparrow^{i+1}(c/)])[\Uparrow^{i}(s)] & & (\lambda a[\Uparrow^{i+2}(s)])[\Uparrow^{i}(c[s]/)] \\
\downarrow{\scriptstyle lam} & & \downarrow{\scriptstyle lam} \\
(\lambda a[\Uparrow^{i+1}(c/)][\Uparrow^{i+1}(s)]) & \xrightarrow[c2]{} & (\lambda a[\Uparrow^{i+2}(s)][\Uparrow^{i+1}(c[s]/)])
\end{array}
$$

- (FVarLift) with (c3) ($i \geq 1$):

$$
\begin{array}{ccc}
1[\Uparrow^{i}(\uparrow)][\Uparrow^{i+1}(s)] & \xrightarrow[c3]{} & 1[\Uparrow^{i}(s)][\Uparrow^{i}(\uparrow)] \\
\downarrow{\scriptstyle FVarLift} & & \downarrow{\scriptstyle FVarLift} \\
1[\Uparrow^{i+1}(s)] & \xrightarrow[FVarLift]{} & 1
\end{array}
$$

- (c1) with (c2) ($i \geq 0$):

  Two possible overlaps:

  1. $$
     \begin{array}{ccc}
     a[\Uparrow^{i}(\uparrow)][\Uparrow^{i}(b/)][\Uparrow^{i}(s)] & \xrightarrow[c2]{} & a[\Uparrow^{i}(\uparrow)][\Uparrow^{i+1}(s)][\Uparrow^{i}(b[s]/)] \\
     \downarrow{\scriptstyle c1} & & \downarrow{\scriptstyle c3} \\
     a[\Uparrow^{i}(s)] & \xleftarrow{c1} & a[\Uparrow^{i}(s)][\Uparrow^{i}(\uparrow)][\Uparrow^{i}(b[s]/)]
     \end{array}
     $$

2. $a[\Uparrow^i (b/)][\Uparrow^{i+j} (\uparrow)][\Uparrow^{i+j} (c/)] \xrightarrow[c2]{} a[\Uparrow^{i+1+j} (\uparrow)][\Uparrow^i (b[\Uparrow^j (\uparrow)]/)][\Uparrow^{i+j} (c/)]$

$c1 \downarrow$ $\qquad$ $c2 \downarrow$

$a[\Uparrow^i (b/)] \qquad a[\Uparrow^{i+j+1} (\uparrow)][\Uparrow^{i+j+1} (c/)][\Uparrow^i (b[\Uparrow^j (\uparrow)][\Uparrow^j (c/)])/]$

$\uparrow c1 \qquad\qquad c1 \swarrow$

$a[\Uparrow^i (b[\Uparrow^j (\uparrow)][\Uparrow^j (c/)]/)]$

- (RVarLift) with (c3) ($i \geq 1$):

$(n+1)[\Uparrow^i (\uparrow)][\Uparrow^{i+1} (s)] \xrightarrow[c3]{} (n+1)[\Uparrow^i (s)[\Uparrow^i (\uparrow)]$

$RVarLift \downarrow$ $\qquad$ $RvarLift \downarrow$

$n[\Uparrow^{i-1} (\uparrow)][\uparrow][\Uparrow^{i+1} (s)] \qquad n[\Uparrow^{i-1} (s)][\uparrow][\Uparrow^i (\uparrow)]$

$c3 \downarrow$ $\qquad$ $c3 \downarrow$

$n[\Uparrow^{i-1} (\uparrow)][\Uparrow^i (s)][\uparrow] \xrightarrow[c3]{} n[\Uparrow^{i-1} (s)][\Uparrow^{i-1} (\uparrow)][\uparrow]$

- (VarShift) with (c3) ($i = 0$):

$n[\uparrow][\Uparrow (s)] \xrightarrow[c3]{} n[s][\uparrow]$

$VarShift \downarrow \qquad \nearrow RVarLift$

$(n+1)[\Uparrow (s)]$

- (App) with (c3):

$(ab)[\Uparrow^i (\uparrow)][\Uparrow^{i+1} (s)] \xrightarrow[c3]{} (ab)[\Uparrow^i (s)][\Uparrow^i (\uparrow)]$

$app \downarrow$ $\qquad$ $app \downarrow$

$((a[\Uparrow^i (\uparrow)])(b[\Uparrow^i (\uparrow)]))[\Uparrow^{i+1} (s)] \qquad (a[\Uparrow^i (s)]b[\Uparrow^i (s)])[\Uparrow^i (\uparrow)]$

$app \downarrow$ $\qquad$ $app \downarrow$

$(a[\Uparrow^i (\uparrow)][\Uparrow^{i+1} (s)])(b[\Uparrow^i (\uparrow)][\Uparrow^{i+1} (s)]) \xrightarrow[c3]{} (a[\Uparrow^i (s)][\Uparrow^i (\uparrow)])(b[\Uparrow^i (s)][\Uparrow^i (\uparrow)])$

- (Lam) with (c3):

$(\lambda a)[\Uparrow^i (\uparrow)][\Uparrow^{i+1} (s)] \xrightarrow[c3]{} (\lambda a)[\Uparrow^i (s)][\Uparrow^i (\uparrow)]$

$lam \downarrow$ $\qquad$ $lam \downarrow$

$(\lambda a[\Uparrow^{i+1} (\uparrow)])[\Uparrow^{i+1} (s)] \qquad (\lambda a[\Uparrow^{i+1} (s)])[\Uparrow^i (\uparrow)]$

$lam \downarrow$ $\qquad$ $lam \downarrow$

$(\lambda a[\Uparrow^{i+1} (\uparrow)][\Uparrow^{i+2} (s)]) \xrightarrow[c3]{} (\lambda a[\Uparrow^{i+1} (s)][\Uparrow^{i+1} (\uparrow)])$

- (c1) with (c3) ($i \geq 0, j \geq 0$):

$$\begin{array}{ccc} a[\Uparrow^i (\uparrow)][\Uparrow^{i+j+1} (\uparrow)][\Uparrow^{i+j+1} (b/)] & \xrightarrow[c3]{} & a[\Uparrow^i (\Uparrow^j (\uparrow))][\Uparrow^i (\uparrow)][\Uparrow^{i+j+1} (b/)] \\ \downarrow{\scriptstyle c1} & & \downarrow{\scriptstyle c3} \\ a[\Uparrow^i (\uparrow)] & \xleftarrow{c1} & a[\Uparrow^{i+j} (\uparrow)][\Uparrow^{i+j} (b/)][\Uparrow^i (\uparrow)] \end{array}$$

- (c2) with (c3) ($i, j \geq 0$):

  Two possible overlaps:

  1.
$$\begin{array}{ccc} a[\Uparrow^i (b/)][\Uparrow^{i+j} (\uparrow)][\Uparrow^{i+j+1} (s)] & \xrightarrow[c2]{} & a[\Uparrow^{i+j+1} (\uparrow)][\Uparrow^i (b[\Uparrow^j (\uparrow)]/)][\Uparrow^{i+j+1} (s)] \\ \downarrow{\scriptstyle c3} & & \| \\ a[\Uparrow^i (b/)][\Uparrow^{i+j} (s)][\Uparrow^{i+j} (\uparrow)] & & (*) \\ \downarrow{\scriptstyle c2} & & \\ a[\Uparrow^{i+j+1} (s)][\Uparrow^i (b[\Uparrow^j (s)]/)][\Uparrow^{i+j} (\uparrow)] & & \\ \downarrow{\scriptstyle c2} & & \\ a[\Uparrow^{i+j+1} (s)][\Uparrow^{i+j+1} (\uparrow)][\Uparrow^i (b[\Uparrow^j (s)][\Uparrow^j (\uparrow)/])] = (**) & & \end{array}$$

  where

$$(*) \xrightarrow[c2]{} a[\Uparrow^{i+j+1} (\uparrow)][\Uparrow^{i+j+2} (s)][\Uparrow^i (b[\Uparrow^j (\uparrow)][\Uparrow^{j+1} (s)]/)]$$

$$\xrightarrow[c3]{} a[\Uparrow^{i+j+1} (s)][\Uparrow^{i+j+1} (\uparrow)][\Uparrow^i (b[\Uparrow^j (\uparrow)][\Uparrow^{j+1} (s)]/)]$$

$$\xrightarrow[c3]{} a[\Uparrow^{i+j+1} (s)][\Uparrow^{i+j+1} (\uparrow)][\Uparrow^i (b[\Uparrow^j (s)][\Uparrow^j (\uparrow)])/] = (**)$$

  2.
$$\begin{array}{ccc} a[\Uparrow^i (\uparrow)][\Uparrow^{i+j+1} (b/)][\Uparrow^{i+j+1} (s)] & \xrightarrow[c3]{} & a[\Uparrow^{i+j} (b/)][\Uparrow^i (\uparrow)][\Uparrow^{i+j+1} (s)] \\ \downarrow{\scriptstyle c2} & & \downarrow{\scriptstyle c3} \\ a[\Uparrow^i (\uparrow)][\Uparrow^{i+j+2} (s)][\Uparrow^{i+j+1} (b[s]/)] & & a[\Uparrow^{i+j} (b/)][\Uparrow^{i+j} (s)][\Uparrow^i (\uparrow)] \\ \downarrow{\scriptstyle c3} & & \downarrow{\scriptstyle c2} \\ a[\Uparrow^{i+j+1} (s)][\Uparrow^i (\uparrow)][\Uparrow^{i+j+1} (b[s]/)] & \xrightarrow[c3]{} & a[\Uparrow^{i+j+1} (s)][\Uparrow^{i+j} (b[s]/)][\Uparrow^i (\uparrow)] \end{array}$$

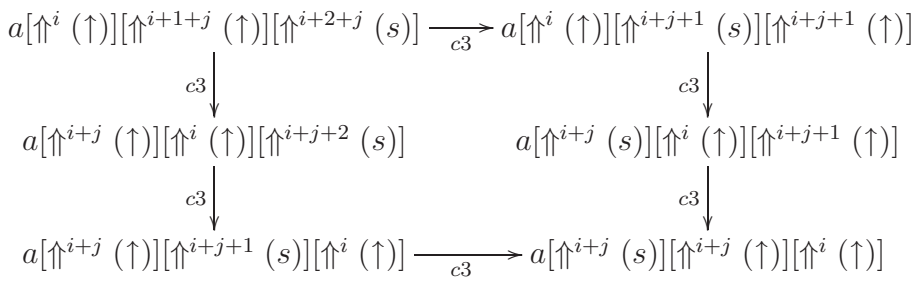
- (c2) with (c2) where $i, j \geq 0$, see Figure 5.1

- (c3) with (c3):

$$
\begin{array}{ccc}
a[\Uparrow^{i}\ (b/)][\Uparrow^{i+j}\ (c/)][\Uparrow^{i+j}\ (s)] & \xrightarrow[c2]{\qquad\qquad} & a[\Uparrow^{i}\ (b/)][\Uparrow^{i+j+1}\ (s)][\Uparrow^{i+j}\ (c[s]/)] \\
\downarrow{\scriptstyle c2} & & \downarrow{\scriptstyle c2} \\
a[\Uparrow^{i+j+1}\ (c/)][\Uparrow^{i}\ (b[\Uparrow^{j}\ (c/)]/)][\Uparrow^{i+j}\ (s)] & & a[\Uparrow^{i+j+2}\ (s)][\Uparrow^{i}\ (b[\Uparrow^{j+1}\ (s)]/)][\Uparrow^{i+j}\ (c[s]/)] \\
\downarrow{\scriptstyle c2} & & \downarrow{\scriptstyle c2} \\
a[\Uparrow^{i+j+1}\ (c/)][\Uparrow^{i+j+1}\ (s)][\Uparrow^{i}\ (b[\Uparrow^{j}\ (c/)][\Uparrow^{j}\ (s)]/)] & & a[\Uparrow^{i+j+2}\ (s)][\Uparrow^{i+j+1}\ (c[s]/)][\Uparrow^{i}\ (b[\Uparrow^{j+1}\ (s)][\Uparrow^{j}\ (c[s]/)]/)] \\
\downarrow{\scriptstyle c2} & & \\
a[\Uparrow^{i+j+2}\ (s)][\Uparrow^{i+j+1}\ (c[s]/)][\Uparrow^{i}\ (b[\Uparrow^{j}\ (c/)][\Uparrow^{j}\ (s)]/)] & &
\end{array}
$$

and it is the case that $b[\Uparrow^{j}\ (c/)][\Uparrow^{j}\ (s)] \underset{c2}{\longrightarrow} b[\Uparrow^{j+1}\ (s)][\Uparrow^{j}\ (c[s]/)]$ therefore the diagram is closed.

Figure 5.1: $\lambda\upsilon_c$ critical pair (c2) with (c2) where $i, j \geq 0$.



□

It is important to note that even when one can check the critical pairs in an automated and convenient way using tools such as CiME (23), no simple tool is likely to be useful for our setting since the terms are being "parameterized" by natural numbers, such as the index i in the substitution $\Uparrow^i(s)$ reflecting its "genericity", actually not part of the syntax[1]. Therefore each one of the above critical pair diagrams has to be considered an infinite family of diagrams.

In a same way we can get WCR for the substitution calculus:

Proposition 5.3.3 *υ_c is WCR for open terms.*

PROOF: Restricting the attention only to the substitution calculus critical pairs.

Other way to prove it is by using Proposition 5.3.2 and the fact that υ_c is a sub-ARS of $\lambda\upsilon_c$.
□

With the present formulation, we are not sure if υ_c is SN, i.e. there is the possibility that composition rules eventually create infinite derivations.

Every closed term a has a υ_c-nf which is a pure term. This is immediate because υ is a SN sub-calculus of υ_c, thus with the υ-rules a term always normalizes.

Let us discuss a possible comparison with λs_e (see the preliminaries). A natural question is if we could transfer our $\lambda\upsilon_c$ formulation to λs-style calculi, by means of a translation from $\lambda\upsilon$ open terms to λs_e, i.e. λs with new rules for open terms (see the preliminaries). It is possible to check that the composition rules are not the result of translating the λs_e composition rules by the standard translations. Nor viceversa, the latter cannot be obtained by backward translation of the former.

We emphasize that the composition rules of λs_e are quite different from the rules that we are using with $\lambda\upsilon$ in this chapter, even when some of them may look similar. As in λs_e our rules are rule schemas because of the supraindices i denoting natural numbers as we previously explained. But one almost immediate advantage of these rules with respect to those of λs_e is that they do not handle conditions given by inequalities, so this offers more clarity in understanding the calculus.

[1] In ELAN (53) it is possible to define several rules by using a variable ranging over a finite set of natural numbers as a parameter, and critical pairs can be produced mechanically, but this also yields finite families of them.

5.4 A calculus with one index

As an application of the previous section, we here show a calculus of explicit substitution à la de Bruijn with 1 as its sole index, somehow in the way $\lambda\sigma$ works. This is feasible due to the presence of composition rules like the ones given in section 5.3. The resulting calculus has a minimal set of 8 rules, without the rules (Shift), (RVar) and (RVarLift).

Definition 5.4.1 *We introduce the $\lambda\upsilon_1$-calculus set of open terms $\Lambda^t_{\upsilon 1}$ and its set of open substitutions $\Lambda^s_{\upsilon 1}$ given by the following syntax:*

$$\textbf{Open terms} \quad a \quad ::= \quad 1 \mid X \mid (a\ a) \mid (\lambda a) \mid a[s]$$
$$\textbf{Open substitutions} \quad s \quad ::= \quad x \mid a/ \mid \uparrow \mid \Uparrow(s)$$

where X denotes term meta-variables and x denotes substitution meta-variables; and the following rule schemas:

$$(\lambda a)b \to a[b/] \qquad (\mathrm{Beta}_1)$$
$$1[a/] \to a \qquad (\mathrm{FVar}_1)$$
$$1[\Uparrow(s)] \to 1 \qquad (\mathrm{FVarLift}_1)$$
$$(ab)[s] \to a[s]b[s] \qquad (\mathrm{App}_1)$$
$$(\lambda a)[s] \to \lambda(a[\Uparrow(s)]) \qquad (\mathrm{Lam}_1)$$
$$a[\Uparrow^i(\uparrow)][\Uparrow^i(b/)] \to a \quad i \geq 0 \qquad (\mathrm{c1}_1)$$
$$a[\Uparrow^i(b/)][\Uparrow^i(s)] \to a[\Uparrow^{i+1}(s)][\Uparrow^i(b[s]/)] \quad i \geq 0 \qquad (\mathrm{c2}_1)$$
$$a[\Uparrow^i(\uparrow)][\Uparrow^{i+1}(s)] \to a[\Uparrow^i(s)][\Uparrow^i(\uparrow)] \quad i \geq 0 \qquad (\mathrm{c3}_1)$$

We call υ_1 to all $\lambda\upsilon_1$-rules except (Beta_1). Let $\Lambda_{\upsilon 1} = \Lambda^t_{\upsilon 1} \cup \Lambda^s_{\upsilon 1}$.

The idea is to represent indices greater than 1 with $1[\uparrow]...[\uparrow]$, to give the translation below. Remark that, in order to have a useful rewriting system, the composition rules become necessary to close diagrams, i.e. one cannot leave them out. For instance, a simulation of the simple reduction $(\lambda 2)b \xrightarrow{*}_{\lambda\upsilon} 1$ would start with the step $(\lambda 1[\uparrow])b \to_{\lambda\upsilon_1} 1[\uparrow][b/]$ and, if one does not consider the $(\mathrm{c1}_1)$-rule, the last term would be a normal form.

Note that $\lambda\upsilon_1$ is a sub-calculus of $\lambda\upsilon_c$ and υ_1 is a sub-calculus of υ_c, as stated by the

Lemma 5.4.2 *Let $a, b \in \Lambda^t_{\upsilon 1}$ be open terms.*

1. *If $a \to_{\upsilon_1} b$, then $a \to_{\upsilon_c} b$.*
2. *If $a \to_{\lambda\upsilon_1} b$, then $a \to_{\lambda\upsilon_c} b$.*

PROOF: Both items are immediate by induction on a using rule-by-rule analysis. □

Definition 5.4.3 *We define the following translation from Λ_v open terms to Λ_{v1} open terms:*

$$t(1) = 1$$
$$t(n+1) = 1\underbrace{[\uparrow]\dots[\uparrow]}_{n}$$
$$t(\lambda a) = \lambda t(a)$$
$$t(ab) = t(a)t(b)$$
$$t(a[s]) = t(a)[t(s)]$$
$$t(X) = X$$
$$t(a/) = t(a)/$$
$$t(\uparrow) = \uparrow$$
$$t(\Uparrow(s)) = \Uparrow(t(s))$$
$$t(x) = x$$

Remark 5.4.4 *t is clearly an onto non-injective application (both considering its restriction to the subset of closed terms as well as with respect to the set of open terms).*

Lemma 5.4.5 *For every $a \in \Lambda_{v1}$ $t(a) = a$.*

PROOF: By induction on a. □

The following Proposition shows that λv_1 simulates (in 0 or more steps) λv_c (in particular, it simulates λv).

Proposition 5.4.6 *Let $a, b \in \Lambda_v$ be open terms. If $a \to_{\lambda v_c} b$, then $t(a) \xrightarrow{*}_{\lambda v_1} t(b)$.*

PROOF: By induction on a. The (RVar)-rule is simulated by the ($c1_1$)- rule. The (FVarLift)-rule is simulated by iteration of the ($c3_1$)-rule. The (VarShift)-rule is simulated trivially in 0 steps. All composition rules are simulated straightforwardly since they are the same, and the other rules and the inductive cases present no problem. □

Lemma 5.4.7 *Let $a \in \Lambda_v$ be a v-nf open term. Then $t(a)$ is a v_1-nf.*

PROOF: Noting that terms of the form $1[\uparrow]\dots[\uparrow]$ do not contain ($c1_1$)-, ($c2_1$)- nor ($c3_1$)-redexes. □

We have

Corollary 5.4.8 *1. v_1 is WCR for open terms.*

2. λv_1 is WCR for open terms.

PROOF: Consequence of the results in section 5.3 about λv_c, Lemma 5.4.2, Proposition 5.4.6 and Lemma 5.4.7. □

5.5 Conclusion and future work

In this chapter we have studied and established the weak confluence on open terms of a $\lambda\upsilon$-calculus with composition-like rules.

The main feature of this result is that we gain weak confluence on open terms in a calculus in a different style to λs and having many of its (good) properties. Also, it is a calculus derived from $\lambda\upsilon$, thus having the same style and minimal rules, using rule schemas (a feature which does not differ from λs_e). This calculus with composition rules has a set of terms which can be shrinked to a smaller set by using only one de Bruijn index, for which an appropriate set of rules can be defined to simulate the parent calculus. This indicates that the composition rules are powerful.

Different things can happen according to the presence of the various substitution constructs. We found that each calculus may have a different behavior from this point of view, a fact that suggests to study them comparatively. It is known that for instance $\lambda\sigma$ loses PSN, principally because of the presence of substitution composition. Thus it would be interesting to explore a general relation between these families of functions and the properties of the calculus: SN of the substitution calculus, PSN, confluence and other.

Future work includes to study SN of the substitution calculus υ_c, and also studying typing systems and PSN for $\lambda\upsilon_c$ and $\lambda\upsilon_1$. Other natural continuation could be to characterize the various substitution functions (and their structures) in different calculi of substitutions. These structures will tell us about the expressive power of the calculi, a notion which is not completely formalized at present.

5.6 Appendix. CINNI-style calculus with composition

In this section we will consider the Calculus of Indexed Names and Named Indices (CINNI) (79). CINNI style of rewriting is greatly inspired in $\lambda\upsilon$, and it introduces the benefit of using both names and de Bruijn indices with the goal of constituting a general theory of higher order rewriting through specifically parameterized substitution calculi. It is sufficiently generic in the sense that it can be instantiated for a wide range of object languages, thus being a particularly interesting subject of study as the author states in (79).

Definition 5.6.1 *Given an underlying first order language $\mathcal{L}$ of function symbols equipped with an arity, the CINNI calculus over $\mathcal{L}$ is given by the following syntax*[1]*:*

$$\begin{array}{ll} \textbf{Terms} & a ::= X_m \mid f(a_1, ...a_n) \mid a[s] \quad \textit{where } m, n \geq 0 \\ \textbf{Substitutions} & s ::= [X = a] \mid \uparrow_X \mid \Uparrow (s)_X \end{array}$$

[1] We use a different closure notation w.r.t. (79): in the closures we precede substitution by the affected term.

where X ranges over a denumerably infinite set of variables; and by the following rules:

$$X_0[X = a] \to a \tag{1c}$$

$$X_{n+1}[X = a] \to X_n \tag{2c}$$

$$Y_n[X = a] \to Y_n \quad if\ X \neq Y \tag{2c'}$$

$$X_0[\Uparrow (s)_X] \to X_0 \tag{3c}$$

$$X_{n+1}[\Uparrow (s)_X] \to X_n[s][\uparrow_X] \tag{4c}$$

$$Y_n[\Uparrow (s)_X] \to Y_n[s][\uparrow_X] \quad if\ X \neq Y \tag{4c'}$$

$$X_n[\uparrow_X] \to X_{n+1} \tag{5c}$$

$$Y_n[\uparrow_X] \to Y_n \quad if\ X \neq Y \tag{5c'}$$

Furthermore, for each syntactic constructor f of the underlying language $\mathcal{L}$ we add a syntax-specific equation in the form of a rewriting rule:

$$f(P_1, \ldots, P_n)[s] \to f(P_1[s_1], \ldots, P_n[s_n]) \tag{fc}$$

where the rhs is a term headed by f but in which the arguments have as closures the substitutions s_i which are basically chains of lifts over s, in possibly different ways which solely depend on f. For example, in $CINNI_c$ (for the λ-calculus, to be defined below) the λ binder is an arity 1 symbol which is always lifted at level 1 in its unique argument, and the application is an arity 2 symbol for which arguments are not lifted at all.

When instantiating this rule appropriately, that is, when using different function symbols with arities and closure propagation criteria as described above, we obtain the (App)- and (Lam)-rules in $\lambda\upsilon$. Thus $\lambda\upsilon$ is clearly a sub-calculus of CINNI, and in this manner CINNI latter represents a big family of extensions of $\lambda\upsilon$.

Now, for s a substitution, $\overline{W} = W_1, \ldots, W_n$ a sequence of n meta-variables and $\overline{Z} = Z_1, \ldots, Z_m$ another sequence of m meta-variables, we use the notation $|Z|_X$ = the number of occurrences of X in the list Z,
also $\Uparrow (s)_{\overline{W}} = \Uparrow (\Uparrow (\ldots \Uparrow (s)_{W_n} \ldots)_{W_2})_{W_1}$
and for concatenation we write
$\overline{W}, X = W_1, \ldots, W_n, X$
$\overline{W}, \overline{Z} = W_1, \ldots, W_n, Z_1, \ldots, Z_m$.

We now introduce a completion of CINNI.

Definition 5.6.2 *The $CINNI_c$ (for the λ-calculus) is defined by the following rules:*

- *All CINNI rules*
- *with regard to elements $f \in \mathcal{L}$, the rules (BetaC), (AppC) and (LamC) below:*

$$(\lambda X.a)b \to a[X = b] \tag{BetaC}$$
$$(ab)[s] \to a[s]b[s] \tag{AppC}$$
$$(\lambda X.a)[s] \to \lambda X.(a[\Uparrow (s)_X]) \tag{LamC}$$

- *the following composition rules:*

$$a[\Uparrow (\uparrow_X)_{\overline{W}}][\Uparrow (X = b)_{\overline{W}}] \to a \tag{c1c}$$
$$a[\Uparrow (X = b)_{\overline{W}}][\Uparrow (s)_{\overline{W}}] \to a[\Uparrow (s)_{\overline{W},X}][\Uparrow (X = s(b))_{\overline{W}}] \tag{c2c}$$
$$a[\Uparrow (\uparrow_X)_{\overline{W}}][\Uparrow (s)_{\overline{W},X}] \to a[\Uparrow (s)_{\overline{W}}][\Uparrow (\uparrow_X)_{\overline{W}}] \tag{c3c}$$

The weak confluence of the $CINNI_c$-calculus on open terms can be proved in the same way as before by analyzing the critical pairs (we omit the various diagrams).

5.7 Appendix. Adding identity to $\lambda\upsilon$

Since we saw in Proposition 5.2.17 that there are no generic identities in $\lambda\upsilon$, we will see below the consequence of adding a (full) identity to this calculus. The interesting thing is that it will not lose its main properties. First we propose to enrich the syntax with another substitution: id, and a new rule can be added accordingly. In this way we introduce the simply typed $\lambda\upsilon^{id}$-calculus term syntax as follows.

Definition 5.7.1

Terms	$a ::= n \mid (a\ a) \mid (\lambda\ a) \mid a[s] \quad$ *where* $n \in \mathbb{N}$
Substitutions	$s ::= id \mid a\ / \mid \uparrow \mid \Uparrow (s)$

with the usual notation.

The rules will consist of the regular $\lambda\upsilon$-rules to which an identity rule is to be added. We first consider the following one:

$$(id_1) \quad n[id] \ \to \ n$$

Note that the effect is not the same as adding the (more efficient) rule

$$(id_2) \quad a[id] \ \to \ a$$

since it can be applied to every term.

Let us call $\lambda\upsilon^{id_2}$ the full calculus adding this rule, and υ^{id_2} the substitution calculus adding this rule.

In order to justify what comes next, we show that $\Uparrow^m (id)$ will also behave as an identity for pure terms.

Lemma 5.7.2 *For all $a \in \Lambda$, $m \in I\!N_0$, $a[\Uparrow^m (id)] \xrightarrow{+}_{\upsilon^{id_2}} a$.*

PROOF: We first prove that for all $n \in I\!N$, $m \in I\!N_0$, $n[\Uparrow^m (id)] \xrightarrow{+}_{\upsilon^{id_2}} n$. This is simple considering $n = 1$ and $n > 1$ separately. Then use induction on a to prove $a[\Uparrow^m (id)] \xrightarrow{+}_{\upsilon^{id_2}} a$. □

Note that Lemma 5.7.2 holds for pure terms and cannot be extended to all $\Lambda\upsilon$. More precisely, if a is a closure then it might not be the case that $a[\Uparrow^m (id)] \xrightarrow{+}_{\upsilon^{id_2}} a$. One may have more than one way to tackle this drawback. One of them is considering a weaker condition: $a[\Uparrow^m (id)] =_\upsilon a$. Then to propose a (restricted) composition rule for $\lambda\upsilon$. Recall that $a[b/][s] =_\upsilon a[\Uparrow (s)][b[s]/]$ can be proved for $\lambda\upsilon$, so we can define the following (restricted) composition rule:

$$(c) \quad a[b/][s] \quad \to \quad a[\Uparrow (s)][b[s]/]$$

But in this case Lemma 5.7.2 can be extended to $\Lambda\upsilon$ only for closures of the form $a[b/]$, since $a[s][\Uparrow^m (id)] \to_c a[\Uparrow^{m+1} (id)][s] \xrightarrow{+}_{\upsilon^{id_2}} a[s]$ by IH (in the proof of Lemma 5.7.2).

One way to solve this is adding composition rules (see next section). The other is to add a more powerful identity.

Therefore adding an identity to this $\lambda\upsilon$ with composition as in rule (id_1) or (id_2) would not have the same effect than adding the more general (and efficient) rule

$$(id) \quad a[\Uparrow^m (id)] \quad \to \quad a \qquad (m \geq 0)$$

as we shall see.

Definition 5.7.3 *We call υ with identity, or υ^{id}, the υ-calculus together with the above (id)-rule schema. Let also $\lambda\upsilon^{id} = \upsilon^{id} + (Beta)$. We call υ^{idp} the calculus restricting the application of the* (id)*-rule to pure terms, and we denote with $\Lambda\upsilon^t_{id}$ the set of terms and with $\Lambda\upsilon^s_{id}$ the set of substitutions.*

Even if we restrict the application of rule *id* to pure terms, this new rule makes sense, since this calculus has the required properties. We state the

Proposition 5.7.4 (Soundness of υ^{idp}) . *The following is valid: For all $a \in \Lambda\upsilon$, $m \in I\!N_0$, $a[\Uparrow^m (id)] =_{\upsilon^{idp}} a$.*

PROOF: Use that $\upsilon(a)[\Uparrow^m (id)] \stackrel{*}{\rightarrow}_{\upsilon^{idp}} \upsilon(a)$ and $a \stackrel{*}{\rightarrow}_{\upsilon} \upsilon(a)$. □

Having said this, for the rest of this section we consider υ^{id} as having the full identity rule given above, i.e. for all $\Lambda\upsilon$ terms.

In the last part of this section we study the υ^{id}-calculus.

We define the following *identity erasure* function over $\Lambda\upsilon^{id}$ terms and substitutions:

$$\begin{array}{lcl} e(n) & = & n \\ e(\lambda a) & = & \lambda e(a) \\ e(ab) & = & e(a)e(b) \\ e(a[\Uparrow^m (id)]) & = & e(a) \\ e(a[s]) & = & e(a)[e(s)] \quad \text{if } s \neq \Uparrow^m (id) \\ e(\Uparrow (s)) & = & \Uparrow (e(s)) \\ e(\uparrow) & = & \uparrow \\ e(a/) & = & e(a)/ \end{array}$$

or equivalently:

$$\begin{array}{lcl} e(n) & = & n \\ e(\lambda a) & = & \lambda e(a) \\ e(ab) & = & e(a)e(b) \\ e(a[\Uparrow^m (id)]) & = & e(a) \\ e(a[\Uparrow^m (b/)]) & = & e(a)[\Uparrow^m (e(b)/)] \\ e(a[\Uparrow^m (\uparrow)]) & = & e(a)[\Uparrow^m (\uparrow)] \end{array}$$

Lemma 5.7.5 *For all $w \in \Lambda\upsilon^{id}$ term or substitution, $w \stackrel{*}{\rightarrow}_{id} e(w)$*

PROOF: By induction on w. Note that the derivation is empty iff $w = e(w)$ iff $w \in \Lambda\upsilon$. □

We show the following *weak projection*:

Lemma 5.7.6 *Let $a, b \in \Lambda\upsilon^{id}$.*

1. *If $a \stackrel{*}{\rightarrow}_{\upsilon^{id}} b$, then $e(a) =_{\upsilon} e(b)$.*
2. *If $a \rightarrow_{Beta} b$, then $e(a) \rightarrow_{Beta} e(b)$.*
3. *If $a \stackrel{*}{\rightarrow}_{\lambda\upsilon^{id}} b$, then $e(a) =_{\lambda\upsilon} e(b)$.*

PROOF:

1. We prove first that if $a \rightarrow_{\upsilon^{id}} b$, then $e(a) =_{\upsilon} e(b)$, by induction on a and mechanically checking each υ^{id}-rule. Then we use induction on the length of the derivation $a \stackrel{*}{\rightarrow}_{\upsilon^{id}} b$.
2. We prove that if $a \rightarrow_{Beta} b$, then $e(a) \rightarrow_{Beta} e(b)$ by induction on a.
3. By (1) and (2).

□

Actual projection, i.e. if $a \to_{\upsilon^{id}} b$ then $e(a) \stackrel{*}{\to}_{\upsilon} e(b)$, is not valid, taking for instance $a = (n+1)[\Uparrow (id)]$, and $b = n[id][\uparrow]$, then $a \to_{\upsilon^{id}} b$ and obviously it is not the case that $n+1 \stackrel{*}{\to}_{\upsilon} n[\uparrow]$[1]. This will slightly modify the following proof with respect to other known confluence proofs.

Proposition 5.7.7 (Confluence of $\lambda\upsilon^{id}$ and υ^{id}) *1. υ^{id} is confluent.*

2. $\lambda\upsilon^{id}$ is confluent.

PROOF:

1. Let $a, b, c \in \Lambda\upsilon^{id}$ such that $a \stackrel{*}{\to}_{\upsilon^{id}} b$ and $a \stackrel{*}{\to}_{\upsilon^{id}} c$. By Lemma 5.7.5, $a \stackrel{*}{\to}_{id} e(a)$, $b \stackrel{*}{\to}_{id} e(b)$ and $c \stackrel{*}{\to}_{id} e(c)$. By Lemma 5.7.6(1), $e(a) =_{\upsilon} e(b)$ and $e(a) =_{\upsilon} e(c)$, thus $e(b) =_{\upsilon} e(c)$. Therefore, by the confluence of υ, there exists d such that $e(b) \stackrel{*}{\to}_{\upsilon} d$ and $e(c) \stackrel{*}{\to}_{\upsilon} d$ and we are done.
2. Analogous to the previous item, using Lemmas 5.7.5 and 5.7.6(3) and the fact that $\lambda\upsilon$ is confluent.

□

Remark that the *(id)*-rule is itself SN since it is size-decreasing for all terms with respect to $|\bullet|$. We can show that SN holds for υ^{id}-calculus.

Proposition 5.7.8 *Strong Normalization holds for υ^{id}-calculus.*

PROOF: We define the function $h : \Lambda\upsilon^{id} \to I\!N$, resembling the definition in (60) but with the addition of the last clause:

$$\begin{array}{lcl} h(n) & = & n+1 \\ h(ab) & = & h(a)+h(b)+1 \\ h(\lambda a) & = & h(a)+1 \\ h(a[s]) & = & h(a)h(s) \\ h(a/) & = & h(a) \\ h(\uparrow) & = & 2 \\ h(\Uparrow (s)) & = & h(s) \\ h(id) & = & 2 \end{array}$$

It is easy to check that whenever $a \to_{\upsilon^{id}} b$, $h(a) > h(b)$. For the new case, since $h(\Uparrow^n (id)) = h(id) = 2$ (by induction on n) then $h(a[\Uparrow^n (id)]) = 2h(a) > h(a)$ for all a. □

Corollary 5.7.9 *PSN holds for $\lambda\upsilon^{id}$*

PROOF: If $a \in \Lambda$ is a SN term, it can be proved by induction on a that no id-redex is created from a after any $\lambda\upsilon^{id}$-step. Then an infinite $\lambda\upsilon^{id}$-derivation from a would be an infinite $\lambda\upsilon$-derivation starting from a, thus by PSN of $\lambda\upsilon$ (13; 60) a is SN with respect to $\lambda\upsilon^{id}$. □

[1] In fact only the *(RVarLift)*-rule blocks the possibility of projection.


Just place your quasi-order.
Waiter ... can you be more efficient? Do you have an idea about complexity ?
Well ... The life reduced me to this. Each time I apply for a new job, nothing changes. But don't argue with me ... I need no argument.
Waiter ... don't you have PURE water here ?
Sure ... types are not included.
I have a decision problem ... maybe I'll have a tea, or a τ
I can give you CafeOBJ, with some syntactic sugar (it is not strong ... ly normalizing)
I see ... TO FIX or NOT TO FIX... That is the POINT
You know I have a crisis of I (dentity) I am ... (Y I). Why me?
Böhm out of here
" Rules must be obeyed " – ABBA, The winner takes it all
MAIN MENU

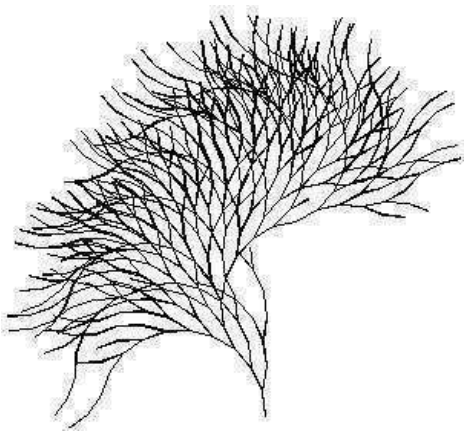


Figure 5.2: $SS(SSS(SS))(SSS(S(SSSS)))$ after 33 left-most steps

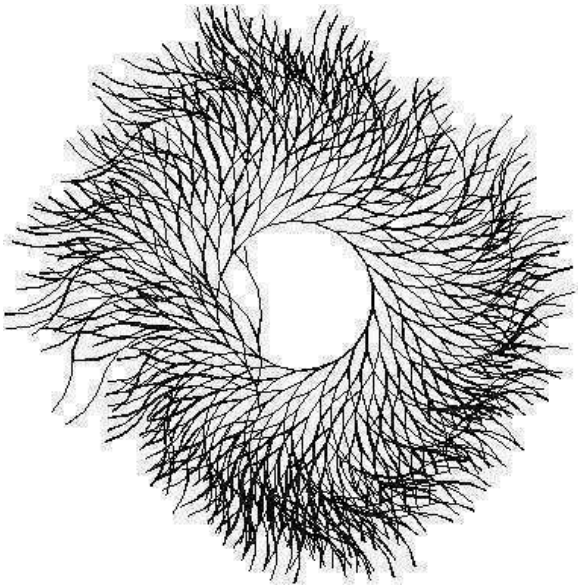


Figure 5.3: $SS((SSS)(SS))(SSS(S((SSS)S)))$ after 73 left-most steps

Chapter 6

The Weak Normalization of the Simply Typed λs_e-calculus

Computer Science is no more about computers than astronomy is about telescopes. – E. W. Dijkstra

In mathematics you don't understand things. You just get used to them. – J. von Neumann

ABSTRACT In this chapter, we show the weak normalization (WN) of the simply-typed λs_e-calculus with open terms where abstractions are decorated with types, and meta-variables, de Bruijn indices and updating operators are decorated with environments. We show a proof of WN using the $\lambda\omega_e$-calculus, a calculus isomorphic to λs_e. This proof is strongly influenced by Goubault-Larrecq's proof of WN for the $\lambda\sigma$-calculus but with subtle differences which show that the two styles require different attention. Furthermore, we give a new calculus $\lambda\omega'_e$ which works like λs_e but which is closer to $\lambda\sigma$ than $\lambda\omega_e$. For both $\lambda\omega_e$ and $\lambda\omega'_e$ we prove WN for typed semi-open terms (i.e. those which allow term variables but no substitution variables), unlike the result of Goubault-Larrecq which covered all $\lambda\sigma$ open terms.

6.1 Introduction

The objective of this chapter is to study the weak normalization of typable terms in a simply typed de Bruijn lambda calculus with explicit substitution, and some variants. We begin by giving here a brief survey of the simply typed versions of the λ-calculus with de Bruijn indices and of λs, λs_e and their respectively isomorphic (in a sense to be defined later) calculi $\lambda\omega$ and $\lambda\omega_e$ as presented in (47). From the point of view of syntax the only difference in those presentations from the type free framework is that abstractions are decorated with types. In

the calculi we study here we also have meta-variables, de Bruijn indices and updating operators decorated with environments. We will present these extensions in Section 6.2.

A result of weak normalization for typed λs_e with open terms is quite interesting, since on the one hand λs_e is a calculus enjoying most good properties, and on the other it is an open problem whether the s_e-calculus is strongly normalizing. Moreover, we give strategies to calculate normal forms. For the proof of WN we will use the method of Goubault-Larrecq (37) already employed for $\lambda\sigma$, although a non-trivial adaptation is necessary for λs_e as we will see.

This chapter contains the joint work in (7).

6.2 Typing lambda calculi by marking abstractions and operators

In this section we give the typed versions of the calculi we are going to study. We recall the calculus $\lambda\vec{\omega}_e$ from (46), and we remark that when we define this calculus as well as $\lambda\vec{s}_e$ they differ from the previously studied versions (see (46; 47)) in that not only abstractions are decorated with types (as in Church-style formulations of typed λ-calculi), but also meta-variables, de Bruijn indices and updating operators are decorated with environments. These new decorations should allow one to type each typable term in a unique way, in the sense that given a typable term, there is a unique environment and a unique type for it. This will allow us to talk about *the type* of a term. For instance, the type of $n_{A_1\dots A_m}$ will be A_n in the environment $A_1 \dots A_m$ (for $m \geq n$), whereas undecorated terms would have a type depending on the given environment. Nevertheless we remark that all the results to follow also hold when erasing all type decorations.

6.2.1 The simply typed λs and λs_e-calculi

Definition 6.2.1 *The set of* ground simply typed λs-terms, *denoted* $\Lambda\vec{s}$, *and the set of* open simply typed λs-terms, *denoted* $\Lambda\vec{s}_{op}$ *are given as follows :*

$$\begin{array}{lll} \textbf{Ground Terms} & a & ::= \; n_{\mathcal{E}} \mid (a\,a) \mid (\lambda\mathcal{T}.a) \mid a\,\sigma^j a \mid \varphi_k^{i,\mathcal{E}} a \\ \textbf{Open Terms} & a & ::= \; X_{\mathcal{E};\mathcal{T}} \mid n_{\mathcal{E}} \mid (a\,a) \mid (\lambda\mathcal{T}.a) \mid a\,\sigma^j a \mid \varphi_k^{i,\mathcal{E}} a \end{array}$$

where $i, j, n \geq 1$, $k \geq 0$ *and* X *ranges over* $\mathbf{V}$ *a denumerably infinite set of variables with pairs of environments and types as subscripts, denoted as* $X_{E,T}$, $Y_{E,T}$, *etc. For each* $\mathtt{n} \in I\!N$, *we assume that* $\mathtt{n}_E$ *can be formed only when the length of* E *is greater than or equal to* n. *Sometimes, when no ambiguity could arise, the subscripts will be omitted. We call* pure terms *the terms that do not contain meta-variables,* σ- *or* φ-*operators.*

Definition 6.2.2 *1. A* closure *is a term of the form* $a\,\sigma^j b$. Compatibility *on* Λs *is extended by adding: if* $a \to b$ *then* $a\,\sigma^j c \to b\,\sigma^j c$, $c\,\sigma^j a \to c\,\sigma^j b$ *and* $\varphi_k^i a \to \varphi_k^i b$.

$$
\begin{array}{lrcl}
\vec{\sigma}\text{-}gen & (\lambda A.a)\, b & \longrightarrow & a\,\sigma^1\, b \\
\vec{\sigma}\text{-}\lambda\text{-}tr & (\lambda A.a)\,\sigma^j b & \longrightarrow & \lambda A.(a\,\sigma^{j+1}\, b) \\
\vec{\sigma}\text{-}app\text{-}tr & (a_1\, a_2)\,\sigma^j b & \longrightarrow & (a_1\,\sigma^j b)\,(a_2\,\sigma^j b) \\
\vec{\sigma}\text{-}des & \mathtt{n}_{E_{<j},B,E_{\geq j}}\,\sigma^j b & \longrightarrow & \begin{cases} \mathtt{n}-\mathtt{1}_E & \text{if } n>j \\ \varphi_0^{j,E}\, b & \text{if } n=j \\ \mathtt{n}_E & \text{if } n<j \end{cases} \\
\vec{\varphi}\text{-}\lambda\text{-}tr & \varphi_k^{i,E}(\lambda A.a) & \longrightarrow & \lambda A.(\varphi_{k+1}^{i,E}\, a) \\
\vec{\varphi}\text{-}app\text{-}tr & \varphi_k^{i,E}(a_1\, a_2) & \longrightarrow & (\varphi_k^{i,E}\, a_1)\,(\varphi_k^{i,E}\, a_2) \\
\vec{\varphi}\text{-}des & \varphi_k^{i,E_{>k}}\,\mathtt{n}_{E_{\leq k},E_{\geq k+i}} & \longrightarrow & \begin{cases} \mathtt{n}+\mathtt{i}-\mathtt{1}_E & \text{if } n>k \\ \mathtt{n}_E & \text{if } n\leq k \end{cases}
\end{array}
$$

Figure 6.1: The rewriting rules of the simply typed λs-calculus

2. *The simply typed* λs-calculus *is given by the rewriting rules in Figure 6.1, i.e. its reduction relation is the smallest compatible relation generated by those rules. We use* $\lambda\vec{s}$ *to denote this set of rules. The* $\vec{s}$*-calculus is the rewriting system given by the set of rules* $\vec{s} = \lambda\vec{s} \setminus \{\vec{\sigma}\text{-}gen\}$*. The typing rules are given by the typing system* $\mathbf{L}\,\vec{\mathbf{s1}}$ *of Figure 6.2. We will use the symbol* $\vdash_{\mathbf{Ls1}}$ *for this typing relation.*

3. *We say that* $a \in \Lambda\vec{s}$ *is a* well typed *term, or* typed *for short, if there exists an environment* E *and a type* A *such that* $E \vdash_{\mathbf{Ls1}} a : A$*.*

Definition 6.2.3 *1. The set of rules* $\lambda\vec{s}_e$ *is obtained by adding the rules in Figure 6.3 to*

$$
\begin{array}{llll}
(\mathbf{L}\,\vec{\mathbf{s1}} - var) & A, E \vdash \mathtt{1}_{A,E} : A & (\mathbf{L}\,\vec{\mathbf{s1}} - \lambda) & \dfrac{A, E \vdash b : B}{E \vdash \lambda A.b : A \to B} \\
(\mathbf{L}\,\vec{\mathbf{s1}} - varn) & \dfrac{E \vdash \mathtt{n}_E : B}{A, E \vdash \mathtt{n}+\mathtt{1}_{A,E} : B} & (\mathbf{L}\,\vec{\mathbf{s1}} - app) & \dfrac{E \vdash b : A \to B \quad E \vdash a : A}{E \vdash b\, a : B} \\
(\mathbf{L}\,\vec{\mathbf{s1}} - \sigma) & \dfrac{E_{\geq i} \vdash b : B \quad E_{<i}, B, E_{\geq i} \vdash a : A}{E \vdash a\,\sigma^i b : A} & (\mathbf{L}\,\vec{\mathbf{s1}} - \varphi) & \dfrac{E_{\leq k}, E_{\geq k+i} \vdash a : A}{E \vdash \varphi_k^{i,E_{>k}} a : A}
\end{array}
$$

Figure 6.2: The typing rules of the simply typed λs-calculus

$\vec{\sigma}$-$\vec{\sigma}$-tr	$(a\,\sigma^i b)\,\sigma^j\,c$	$\longrightarrow$	$(a\,\sigma^{j+1}\,c)\;\sigma^i\,(b\,\sigma^{j-i+1}\,c)$	if $i \leq j$
$\vec{\sigma}$-$\vec{\varphi}$-$tr\ 1$	$(\varphi_k^{i,E_{<j-k},B,E_{\geq j-k}} a)\,\sigma^j\,b$	$\longrightarrow$	$\varphi_k^{i-1,E}\,a$	if $k < j < k+i$
$\vec{\sigma}$-$\vec{\varphi}$-$tr\ 2$	$(\varphi_k^{i,E_{<j-k},B,E_{\geq j-k}} a)\,\sigma^j\,b$	$\longrightarrow$	$\varphi_k^{i,E}(a\,\sigma^{j-i+1}\,b)$	if $k+i \leq j$
$\vec{\varphi}$-$\vec{\sigma}$-tr	$\varphi_k^{i,E}(a\,\sigma^j\,b)$	$\longrightarrow$	$(\varphi_{k+1}^{i,E} a)\,\sigma^j\,(\varphi_{k+1-j}^{i,E} b)$	if $j \leq k+1$
$\vec{\varphi}$-$\vec{\varphi}$-$tr\ 1$	$\varphi_k^{i,E_{>k-l}}(\varphi_\ell^{j,E_{\leq k-l},E_{\geq k+i-l}} a)$	$\longrightarrow$	$\varphi_\ell^{j,E}\,(\varphi_{k+1-j}^{i,E_{>k-l}}\,a)$	if $l+j \leq k$
$\vec{\varphi}$-$\vec{\varphi}$-$tr\ 2$	$\varphi_k^{i,E_{>k-l}}(\varphi_\ell^{j,E_{\leq k-l},E_{\geq k+i-l}} a)$	$\longrightarrow$	$\varphi_\ell^{j+i-1,E}\,a$	if $l \leq k < l+j$

Figure 6.3: The new rewriting rules of the simply typed λs_e-calculus

the rules of the $\lambda\vec{s}$-calculus given in Figure 6.1. The $\lambda\vec{s}_e$-calculus *is the reduction system $(\Lambda\vec{s}_{op}, \rightarrow_{\lambda\vec{s}_e})$ where $\rightarrow_{\lambda\vec{s}_e}$ is the smallest compatible reduction on $\Lambda\vec{s}_{op}$ generated by the set of rules $\lambda\vec{s}_e$.*

The $\vec{s}_e$-calculus *is the rewriting system generated by the set of rules $\vec{s}_e = \lambda\vec{s}_e \setminus \{\vec{\sigma}\text{-}gen\}$. Remark that the typing rules for $\lambda\vec{s}_e$ are exactly the same as the typing rules for $\lambda\vec{s}$ given in Figure 6.2. We only need to add rules to type meta-variables:*

$$(L\ \vec{s1}\ -Metav) \qquad E \vdash X_{E,A} : A.$$

We further assume that for each context E and type A there are infinitely many meta-variables X, such that $E \vdash X : A$.

2. *We say that $a \in \Lambda\vec{s}_{op}$ is a* well typed *term, or* typed *for short, if there exists an environment E and a type A such that $E \vdash_{\mathbf{Ls1}} a : A$.*

We denote with λs and λs_e the respective untyped calculi, i.e. the calculi where the decorations have been erased.

6.2.2 The simply typed $\lambda\omega$ and $\lambda\omega_e$-calculi

In this section we describe $\lambda\omega$ and $\lambda\omega_e$ where terms are decorated with types and environments. See the discussion about the closure notation and operators in the preliminaries.

Definition 6.2.4 *The set of terms of the $\lambda\vec{\omega}$-calculus, noted $\Lambda\vec{\omega}$, is defined as $\Lambda\,\vec{\omega}^t \cup \Lambda\,\vec{\omega}^s$, where $\Lambda\,\vec{\omega}^t$ (terms) and $\Lambda\,\vec{\omega}^s$ (substitutions) are mutually defined as follows :*

Terms	$a ::= n_{\mathcal{E}} \mid (a\ a) \mid (\lambda\mathcal{T}.a) \mid a[s]_j$	*where $n, j \geq 1$*
Substitutions	$s ::= \uparrow^i_{\mathcal{E}} \mid a\ /$	*where $i \geq 0$*

The set, denoted $\lambda\vec{\omega}$*, of rules of the* $\lambda\vec{\omega}$-calculus *is given in Figure 6.4 . The set of rules of the* $\vec{\omega}$-calculus *is the set* $\vec{\omega}= \lambda\vec{\omega} \setminus\{\vec{\sigma} -gen\}$. Closures *are terms of the form* $a[s]_i$, pure terms *are terms without substitutions, and* compatibility *is defined in the usual way (see the preliminaries).*

The typing system for the $\lambda\vec{\omega}$*-calculus is called* $\mathbf{L}\vec{\omega}\mathbf{1}$*. The rules* $\mathbf{L}\vec{\omega}\mathbf{1}$*-var,* $\mathbf{L}\vec{\omega}\mathbf{1}$*-varn,* $\mathbf{L}\vec{\omega}\mathbf{1}$*-*$\lambda$ *and* $\mathbf{L}\vec{\omega}\mathbf{1}$*-app are exactly the same as* $\mathbf{L}\vec{s}\mathbf{1}$*-var,* $\mathbf{L}\vec{s}\mathbf{1}$*-varn,* $\mathbf{L}\vec{s}\mathbf{1}$*-*$\lambda$ *and* $\mathbf{L}\vec{s}\mathbf{1}$*-app, respectively, as given in Figure 6.2. The new rules are given in Figure 6.5 . We will use the symbol* $\vdash_{\mathbf{L}\vec{\omega}\mathbf{1}}$ *for this typing relation.*

Just like the $\lambda\vec{s}$-calculus, the $\lambda\vec{\omega}$-calculus is not even locally confluent on open terms. By *open terms* in this new syntax we mean terms which admit variables (usually called meta-variables) of sort `term` and of sort `substitution`.

Now, we define formally what we mean by open terms in our new syntax and give the rewriting rules of $\lambda\vec{\omega}_e$:

Definition 6.2.5 *The set of* open terms*, noted* $\Lambda\vec{\omega}_{op}$ *is defined as* $\Lambda\ \vec{\omega}^t_{op} \cup \Lambda\ \vec{\omega}^s_{op}$*, where* $\Lambda\ \vec{\omega}^t_{op}$ *(terms) and* $\Lambda\ \vec{\omega}^s_{op}$ *(substitutions) are mutually defined as follows :*

$$\begin{array}{ll} \textbf{Open Terms} & a ::= X_{\mathcal{E};\mathcal{T}} \mid n_{\mathcal{E}} \mid (a\ a) \mid (\lambda\mathcal{T}.a) \mid a[s]_j \\ \textbf{Open Substitutions} & s ::= x_{\mathcal{E};\mathcal{E}} \mid \uparrow^i_{\mathcal{E}} \mid a\ / \end{array}$$

where $n, j \geq 1$ *and* $i \geq 0$*,* X *ranges over* $\mathbf{V}$ *a denumerably infinite set of term variables denoted with* $X_{E,T}$*,* $Y_{E,T}$*, etc. and* x *ranges over* $\mathbf{W}$ *a denumerably infinite set of substitution variables, denoted with* $x_{E,E'}$*,* $y_{E,E'}$*, etc. Closures, pure terms and compatibility are defined as for* $\lambda\vec{\omega}$*.*

The set, denoted $\lambda\vec{\omega}_e$*, of rules of the* $\lambda\vec{\omega}_e$-calculus *is obtained by adding to the set of rules* $\lambda\vec{\omega}$ *given in Figure 6.4, the rules given in Figure 6.6. The set of rules of the* $\vec{\omega}_e$-calculus *is* $\vec{\omega}_e= \lambda\vec{\omega}_e \setminus\{\vec{\sigma} -gen\}$.

The typing rules for $\lambda\vec{\omega}_e$ *are those of* $\lambda\vec{\omega}$ *together with rules to type meta-variables:*

$$(L\ \vec{s1}\ -MetavT) \quad E \vdash X_{E,A} : A \qquad\qquad (L\ \vec{s1}\ -MetavS) \quad E \vdash x_{E,E'} \triangleright E'$$

We further assume that for each context E *and type* A *there are infinitely many meta-variables* X*, such that* $E \vdash X : A$*. We also assume that for each pair of contexts* E*,* E' *there are infinitely many meta-variables* x*, such that* $E \vdash x \triangleright E'$*.*

We also define $\Lambda\ \vec{\omega}^t_{sop}$ *as the set of* semi-open terms*, i.e. those open terms without substitution variables (it is a proper subset of* $\Lambda\ \vec{\omega}^t_{op}$*), and* $\Lambda\ \vec{\omega}^s_{sop}$ *as the set of* semi-open substitutions*, i.e. those open substitutions without substitution variables (it is a proper subset of* $\Lambda\ \vec{\omega}^s_{op}$*).* $\Lambda\ \vec{\omega}^t_{sop}$ *and* $\Lambda\ \vec{\omega}^s_{sop}$ *are mutually defined as follows:*

$$\begin{array}{ll} \textbf{Semi-open Terms} & a ::= X_{\mathcal{E};\mathcal{T}} \mid n_{\mathcal{E}} \mid (a\ a) \mid (\lambda\mathcal{T}.a) \mid a[s]_j \\ \textbf{Semi-open Substitutions} & s ::= \uparrow^i_{\mathcal{E}} \mid a\ / \end{array}$$

$$
\begin{array}{lrcl}
\vec{\sigma}\text{-}gen & (\lambda A.a)\, b & \longrightarrow & a\,[b/]_1 \\
\vec{\sigma}\text{-}app\text{-}tr & (a\, b)[s]_j & \longrightarrow & (a\,[s]_j)\,(b\,[s]_j) \\
\vec{\sigma}\text{-}\lambda\text{-}tr & (\lambda A.a)[s]_j & \longrightarrow & \lambda A.(a[s]_{j+1}) \\
\vec{\sigma}\text{-}/\text{-}des & \mathtt{n}_{E_{<j},B,E_{\geq j}}[a/]_j & \longrightarrow & \begin{cases} \mathtt{n}-\mathtt{1}_E & \text{if } n > j \\ a[\uparrow_E^{j-1}]_1 & \text{if } n = j \\ \mathtt{n}_E & \text{if } n < j \end{cases} \\
\vec{\sigma}\text{-}\uparrow\text{-}des & \mathtt{n}_{E_{<j},E_{\geq i+j}}[\uparrow^i_{E_{\geq j}}]_j & \longrightarrow & \begin{cases} \mathtt{n}+\mathtt{i}_E & \text{if } n \geq j \\ \mathtt{n}_E & \text{if } n < j \end{cases}
\end{array}
$$

Figure 6.4: The rewriting rules of the simply typed $\lambda\omega$-calculus

$$
\begin{array}{llll}
(\mathbf{L}\vec{\omega}\mathbf{1} - id) & E \vdash \uparrow_E^0 \triangleright E & (\mathbf{L}\vec{\omega}\mathbf{1} - slash) & \dfrac{E \vdash a : A}{E \vdash a/ \triangleright A, E} \\
(\mathbf{L}\vec{\omega}\mathbf{1} - shift) & \dfrac{E \vdash \uparrow_E^i \triangleright E'}{A, E \vdash \uparrow_{A,E}^{i+1} \triangleright E'} & (\mathbf{L}\vec{\omega}\mathbf{1} - clos) & \dfrac{E_{\geq j} \vdash s \triangleright E' \qquad E_{<j}, E' \vdash a : A}{E \vdash a[s]_j : A} \\
(\mathbf{L}\vec{\omega}\mathbf{1} - MetavT) & E \vdash X_{E,A} : A & (\mathbf{L}\vec{\omega}\mathbf{1} - MetavS) & E \vdash x_{E,E'} \triangleright E'
\end{array}
$$

Figure 6.5: The new typing rules of the simply typed $\lambda\omega_e$-calculus

where $n, j \geq 1$ and $i \geq 0$ and X means the same as before. Last, we denote with $\lambda\omega$ and $\lambda\omega_e$ the respective untyped calculi, i.e. the calculi where the decorations have been erased.

In the rest of the chapter unless explicitly stated we will restrict $\lambda\vec{\omega}_e$ to the set $\Lambda\,\vec{\omega}^t_{sop}$.

6.3 The isomorphism

The untyped versions of $\lambda\vec{s}$, $\lambda\vec{s}_e$, $\lambda\vec{\omega}$ and $\lambda\vec{\omega}_e$ are obtained by deleting every type and environment information of terms (cf. (47)). For the untyped calculi, in (47) the authors proved that the term restriction of the $\lambda\vec{\omega}$- and $\lambda\vec{\omega}_e$-calculi on $\Lambda\,\vec{\omega}^t_{sop}$ are "isomorphic" respectively to the $\lambda\vec{s}$- and $\lambda\vec{s}_e$-calculi. In this section, we state that the isomorphism can be adapted for the typed versions of these calculi and furthermore, that the new isomorphism preserves types.

Definition 6.3.1 *The functions $T : \Lambda\vec{s}_{op} \to \Lambda\,\vec{\omega}^t_{sop}$ and $S : \Lambda\,\vec{\omega}^t_{sop} \to \Lambda\vec{s}_{op}$ are defined induc-*

$$
\begin{array}{lrcl}
\vec{\sigma}\text{-}/\text{-}tr & a\,[b/]_k[s]_j & \longrightarrow & a\,[s]_{j+1}[b[s]_{j-k+1}/]_k \quad \text{if} \quad k \le j \\
/\text{-}\uparrow\text{-}tr & a\,[\uparrow^i_{E_{\le j-k},B,E_{>j-k}}]_k[b/]_j & \longrightarrow & \begin{cases} a[b/]_{j-i}[\uparrow^i_E]_k & \text{if} \quad k+i \le j \\ a[\uparrow^{i-1}_E]_k & \text{if} \quad k \le j < k+i \end{cases} \\
\uparrow\text{-}\uparrow\text{-}tr & a\,[\uparrow^i_{E_{\le j-k},E_{>j+l-k}}]_k[\uparrow^l_{E_{>j-k}}]_j & \longrightarrow & \begin{cases} a[\uparrow^l_{E_{>j-k}}]_{j-i}[\uparrow^i_E]_k & \text{if} \quad k+i < j \\ a[\uparrow^{i+l}_E]_k & \text{if} \quad k \le j \le k+i \end{cases}
\end{array}
$$

Figure 6.6: The new rewriting rules of the simply typed $\lambda\omega_e$-calculus

tively:

$$
\begin{array}{ll}
T(X_{E,A}) = X_{E,A} & S(X_{E,A}) = X_{E,A} \\
T(\mathtt{n}_E) = \mathtt{n}_E & S(\mathtt{n}_E) = \mathtt{n}_E \\
T(a\,b) = T(a)T(b) & S(a\,b) = S(a)S(b) \\
T(\lambda A.a) = \lambda A.T(a) & S(\lambda A.a) = \lambda A.S(a) \\
T(a\,\sigma^j b) = T(a)[T(b)/]_j & S(a\,[b/]_j) = S(a)\,\sigma^j S(b) \\
T(\varphi^{i,E}_k a) = T(a)[\uparrow^{i-1}_E]_{k+1} & S(a\,[\uparrow^i_E]_k) = \varphi^{i+1,E}_{k-1}(S(a))
\end{array}
$$

We make an "abus de notation" and use the same names T and S for the trivial restrictions of these functions to ground terms. The context will make it clear which is meant in every case.

Theorem 6.3.2 *The following hold:*

1. *Let $a, b \in \Lambda\vec{s}_{op}$. If $a \to_{\vec{s}} b$ then $T(a) \to_{\vec{\omega}} T(b)$. If $a \to_{\lambda\vec{s}} b$ then $T(a) \to_{\lambda\vec{\omega}} T(b)$.*
2. *Let $a, b \in \Lambda\vec{s}_{op}$. If $a \to_{\vec{s}_e} b$ then $T(a) \to_{\vec{\omega}_e} T(b)$. If $a \to_{\lambda\vec{s}_e} b$ then $T(a) \to_{\lambda\vec{\omega}_e} T(b)$.*
3. *Let $a, b \in \Lambda\,\vec{\omega}^t_{sop}$. If $a \to_{\vec{\omega}} b$ then $S(a) \to_{\vec{s}} S(b)$. If $a \to_{\lambda\vec{\omega}} b$ then $S(a) \to_{\lambda\vec{s}} S(b)$.*
4. *Let $a, b \in \Lambda\,\vec{\omega}^t_{sop}$. If $a \to_{\vec{\omega}_e} b$ then $S(a) \to_{\vec{s}_e} S(b)$. If $a \to_{\lambda\vec{\omega}_e} b$ then $S(a) \to_{\lambda\vec{s}_e} S(b)$.*

PROOF: By induction on a. If the reduction is internal, the induction hypothesis applies; otherwise, the theorem must be checked for each rule. As an example, we illustrate for item 2. the case of reduction at the root with the rules *σ-φ-tr 1* and *σ-φ-tr 2*:
Suppose $k < j < k+i$. Then $(\varphi^i_k a)\sigma^j b \to_{\sigma-\varphi-tr1} \varphi^{i-1}_k a$, and
$T(\varphi^i_k a)\sigma^j b) = T(\varphi^i_k a)[T(b)/]_j$
$= T(a)[\uparrow^{i-1}]_{k+1}[T(b)/]_j \to_{/-\uparrow-tr} T(a)[\uparrow^{i-2}]_{k+1}$ (since $k+i \ge j \ge k+1 > k$)
$= T(\varphi^{i-1}_k a)$.
Suppose $k+i \le j$. Then $(\varphi^i_k a)\sigma^j b \to_{\sigma-\varphi-tr2} \varphi^i_k(a\sigma^{j-i+1}b)$, and
$T((\varphi^i_k a)\sigma^j b) = T(\varphi^i_k a)[T(b)/]_j$

$= T(a)[\uparrow^{i-1}]_{k+1}[T(b)/]_j \to_{/-\uparrow-tr} T(a)[T(b)/]_{j-i+1}[\uparrow^{i-1}]_{k+1}$ (since $k+i \leq j$)
$= T(a\sigma^{j-i+1}b)[\uparrow^{i-1}]_{k+1} = T(\varphi^i_k(a\sigma^{j-i+1}b))$.
The other cases are simpler. □

We verify finally that T and S are in fact inverses of each other.

Theorem 6.3.3 *The following holds: For all $a \in \Lambda\ \vec{\omega}^t_{sop}$, we have $T(S(a)) = a$. For all $a \in \Lambda\vec{s}_{op}$, we have $S(T(a)) = a$.*

Proof: By an easy induction on a. □

Corollary 6.3.4 *$\lambda\vec{s}$ is isomorphic to $\lambda\vec{\omega}$ restricted to $\Lambda\ \vec{\omega}^t$, and $\lambda\vec{s}_e$ is isomorphic to $\lambda\vec{\omega}_e$ restricted to $\Lambda\ \vec{\omega}^t_{sop}$.*

We end this section by stating that the isomorphism preserves types, which will be used to obtain subject reduction for $\lambda\vec{s}$ and $\lambda\vec{s}_e$ from the corresponding results for $\lambda\vec{\omega}$ and $\lambda\vec{\omega}_e$.

Lemma 6.3.5 *Let E be an environment, A a type, $a \in \Lambda\vec{s}_{op}$ and $b \in \Lambda\ \vec{\omega}^t_{sop}$.*

1. *If $E \vdash_{\mathbf{L}\vec{\mathbf{s}}\mathbf{1}} a : A$ then $E \vdash_{\mathbf{L}\vec{\omega}\mathbf{1}} T(a) : A$.*
2. *If $E \vdash_{\mathbf{L}\vec{\omega}\mathbf{1}} b : A$ then $E \vdash_{\mathbf{L}\vec{\mathbf{s}}\mathbf{1}} S(b) : A$.*

Proof: By induction on the structure of a and b, respectively. □

6.4 Subject Reduction

This section is devoted to establish Subject Reduction for our four calculi. We prove first subject reduction for $\lambda\vec{\omega}$ and $\lambda\vec{\omega}_e$ and then we use the isomorphisms given in the previous section to obtain Subject Reduction for $\lambda\vec{s}$ and $\lambda\vec{s}_e$.

Theorem 6.4.1 (Subject Reduction for $\lambda\vec{\omega}$) *Let $a, b \in \Lambda\ \vec{\omega}^t_{op}$ and $s, t \in \Lambda\ \vec{\omega}^s_{op}$.*

1. *If $E \vdash a : A$ and $a \to_{\lambda\vec{\omega}} b$ then $E \vdash b : A$.*
2. *If $E \vdash s \triangleright F$ and $s \to_{\lambda\vec{\omega}} t$ then $E \vdash t \triangleright F$.*

Proof: By simultaneous induction on the structure of a and s. If the reduction is internal it is enough to apply the inductive hypothesis. If the reduction is at the root then each rule must be examined. We check for instance the rule $\vec{\sigma}$-/-*des* for the case $n = j$.

Let us assume $E \vdash \mathtt{n}_{F_{<j},B,F_{\geq j}}[a/]_j : A$. Therefore there exists an environment E' such that $E_{\geq j} \vdash a/ \triangleright E'$ and $E_{<j}, E' \vdash \mathtt{n}_{F_{<j},B,F_{\geq j}} : A$. Hence $E_{<j} = F_{<j}$ and $E' = B, F_{\geq j}$ and therefore $E_{\geq j} = F_{\geq j}$, hence $E = F$. From $E_{\geq j} \vdash a/ \triangleright E'$ we deduce $E_{\geq j} \vdash a : B$ and, since $A =$

$(E_{<j}, B, E_{\geq j})_n$ and $n = j$, we have $A = B$. Therefore, $E_{\geq j} \vdash a : A$ and, because $E \vdash \uparrow_E^{j-1} \triangleright E_{\geq j}$, we can apply the *clos*-rule (remember $E = E_{\geq 1}$ and, by convention, $E_{<1} = nil$) to obtain $E \vdash a[\uparrow_E^{j-1}]_1 : A$. □

Theorem 6.4.2 (Subject Reduction for $\lambda\vec{\omega}_e$) *Let $a, b \in \Lambda\, \vec{\omega}^t_{op}$ and $s, t \in \Lambda\, \vec{\omega}^s_{op}$.*

1. *If $E \vdash a : A$ and $a \to_{\lambda\vec{\omega}_e} b$ then $E \vdash b : A$.*
2. *If $E \vdash s \triangleright F$ and $s \to_{\lambda\vec{\omega}_e} t$ then $E \vdash t \triangleright F$.*

Proof: By simultaneous induction on the structure of a and s. The proof is analogous to the previous proof, only the new rules must be checked now. As an example we study $\vec{\sigma}$-/-*tr*. Assume $E \vdash a[b/]_k[s]_j : A$ and $k \leq j$. Therefore, there exists an environment E' such that

$$E_{\geq j} \vdash s \triangleright E' \tag{6.1}$$

and $E_{<j}, E' \vdash a[b/]_k : A$. Since $k \leq j$, by $\mathbf{L}\vec{\omega}\mathbf{1}$-*clos* there exists an environment E'' such that

$$E_{<k}, E'' \vdash a : A \tag{6.2}$$

and $E_k, \ldots, E_{j-1}, E' \vdash b/ \triangleright E''$. Therefore, $E'' = B, E_k, \ldots, E_{j-1}, E'$ and

$$E_k, \ldots, E_{j-1}, E' \vdash b : B \tag{6.3}$$

Applying the *clos* rule, from 6.1 and 6.2 we get

$$E_{<k}, B, E_{\geq k} \vdash a[s]_{j+1} : A \tag{6.4}$$

and from 6.1 and 6.3, $E_{\geq k} \vdash b[s]_{j-k+1} : B$, and a further application of *slash* gives

$$E_{\geq k} \vdash b[s]_{j-k+1}/ \triangleright B, E_{\geq k} \tag{6.5}$$

Finally, applying *clos* to 6.4 and 6.5, we conclude $E \vdash a[s]_{j+1}[b[s]_{j-k+1}/]_k : A$. □

We use now the translations to prove Subject Reduction for $\lambda\vec{s}$ and $\lambda\vec{s}_e$. Actually, this will only need the above Subject Reduction result restricted to the set $\Lambda\, \vec{\omega}^t_{sop}$.

Theorem 6.4.3 (Subject Reduction for $\lambda\vec{s}$ and $\lambda\vec{s}_e$) *Let $a, b \in \Lambda\vec{s}$ and $c, d \in \Lambda\vec{s}_{op}$.*

1. *If $E \vdash a : A$ and $a \to_{\lambda\vec{s}} b$ then $E \vdash b : A$.*
2. *If $E \vdash c : A$ and $c \to_{\lambda\vec{s}_e} d$ then $E \vdash d : A$.*

Proof: We just check the 1st item (the 2nd is similar). If $E \vdash a : A$ then, by Lemma 6.3.5.1, $E \vdash T(a) : A$. On the other hand, if $a \to_{\lambda\vec{s}} b$ then, by Theorem 6.3.2.2, $T(a) \to_{\lambda\vec{\omega}} T(b)$. Now, by Theorem 6.4.1.1, $E \vdash T(b) : A$, and by Lemma 6.3.5.2, we get $E \vdash S(T(b)) : A$, and we are done because $S(T(b)) = b$, by Theorem 6.3.3. □

6.5 Weak Normalization of $\vec{\omega}_e$

In this section we prove weak normalization for $\vec{\omega}_e$, the calculus of substitutions associated to $\lambda\vec{\omega}_e$, by reducing the problem to the untyped calculus. Weak normalization of $\vec{\omega}_e$ will be needed in the next section to obtain weak normalization of $\lambda\vec{\omega}_e$.

Definition 6.5.1 *We define* type erasure *for $\lambda\vec{\omega}$-terms as follows:*

$$\begin{array}{ll} |X_{E,A}| = X_{E,A} & |x_{E,F}| = x_{E,F} \\ |n_E| = n & |\uparrow^i_E| = \uparrow^i \\ |a\,b| = |a|\,|b| & |a/| = |a|/ \\ |\lambda A.a| = \lambda|a| & |a[s]_j| = |a|[|s|]_j \end{array}$$

Lemma 6.5.2 *Let $a, b \in \Lambda^t_{sop}(\vec{\omega})$ and $s, t \in \Lambda^s_{sop}(\vec{\omega})$.*

1. *If $a \to_{\lambda\vec{\omega}_e} b$ then $|a| \to_{\lambda\omega_e} |b|$.*
2. *If $s \to_{\lambda\vec{\omega}_e} t$ then $|s| \to_{\lambda\omega_e} |t|$.*

Proof: By an easy induction on the structure of terms and substitutions. □

Theorem 6.5.3 *$\vec{\omega}_e$-calculus is weakly normalizing for semi-open terms.*

Proof: In (46) it is shown that every innermost strategy terminates in the s_e-calculus, i.e. the untyped version of $\vec{s}_e$, for open terms. Here, we prove that every innermost strategy must also terminate for $\vec{\omega}_e$ for semi-open terms. The proof is by contradiction. Let us consider an innermost infinite reduction path beginning with:

- a term a, i.e. $a \to a_1 \to \ldots \to a_n \to \ldots$. Now, using the previous lemma and remarking that erasing the types does not change the character of the strategy, we get an innermost infinite derivation in ω_e:
  $|a| \to |a_1| \to \ldots \to |a_n| \to \ldots$
  and then, applying the translation S (cf.(47)) from untyped $\lambda\vec{\omega}$-terms into untyped $\lambda\vec{s}_e$-terms, which does not change the character of the strategy either, we get the innermost infinite s_e-derivation:
  $S(|a|) \to S(|a_1|) \to \ldots \to S(|a_n|) \to \ldots$
  which contradicts the above mentioned result in (46).
- a substitution s, then $s = a/$ (because $s = \uparrow^i_E$ is a normal form) and the infinite reduction must occur within a. Hence, by the previous item, this is also a contradiction.

□

6.6 Soundness and simulation

We have shown that $\vec{\omega}_e$ is WN for semi-open terms. Therefore for every term a in the corresponding language we can define the normal forms $\vec{s}\ (a)$, $\vec{s}_e\ (a)$, $\vec{\omega_e}\ (a)$ as usual. In this section we show that these calculi enjoy the expected soundness and simulation properties with respect to de Bruijn λ-calculus.

These calculi are sound in the following sense:

Proposition 6.6.1 (Soundness of $\lambda\vec{s}$, $\lambda\vec{s}_e$, $\lambda\vec{\omega}$ and $\lambda\vec{\omega}_e$) *The following hold:*

1. *Let $a, b \in \Lambda\vec{s}$. If $a \to_{\vec{\lambda s}} b$ then $\vec{s}\ (a) \twoheadrightarrow_\beta \vec{s}\ (b)$.*
2. *Let $a, b \in \Lambda\vec{s}_{op}$. If $a \to_{\vec{\lambda s_e}} b$ then $\vec{s}_e\ (a) \twoheadrightarrow_\beta \vec{s}_e\ (b)$.*
3. *Let $a, b \in \Lambda\vec{\omega}$. If $a \to_{\vec{\lambda\omega}} b$ then $\vec{\omega}\ (a) \twoheadrightarrow_\beta \vec{\omega}\ (b)$.*
4. *Let $a, b \in \Lambda\vec{\omega}_{sop}$. If $a \to_{\vec{\lambda\omega_e}} b$ then $\vec{\omega_e}\ (a) \twoheadrightarrow_\beta \vec{\omega_e}\ (b)$.*

Proof: All items can be proved by induction on the position where the reduction takes place. See more details in (46). □

Also, they simulate the β-reduction in the de Bruijn λ-calculus with or without open terms. We recall next the definition of the de Bruijn λ-calculus with the addition of meta-variables.

Definition 6.6.2 *We define the de Bruijn open terms, denoted Λ_{op}, by:*

$$\textbf{Open Terms} \quad a ::= X_{\mathcal{E};\mathcal{T}} \mid n_{\mathcal{E}} \mid (a\ a) \mid (\lambda\mathcal{T}.a)$$

where again X ranges over a denumerably infinite set of variables $\mathbf{V}$. The de Bruijn λ-calculus on open terms has as its only rule β-reduction, which is the smallest compatible reduction on Λ_{op} based on the schema in Definition 1.5.1. As before we may assume that for each context E and type A there are infinitely many meta-variables X, such that $E \vdash X : A$. The typing rules are the usual ones plus the term meta-variable typing rule as given before.

Proposition 6.6.3 (Simulation of β-reduction for $\lambda\vec{s}$, $\lambda\vec{s}_e$, $\lambda\vec{\omega}$ and $\lambda\vec{\omega}_e$) 1. *Let $a, b \in \Lambda$ be typed terms. If $a \to_\beta b$, then the following hold:*

   (a) *there exists $c \in \vec{\Lambda s}$ such that $a \to_{\vec{\sigma}-gen} c \twoheadrightarrow_{\vec{s}} b$.*

   (b) *there exists $c \in \vec{\Lambda\omega}$ such that $a \to_{\vec{\sigma}-gen} c \twoheadrightarrow_{\vec{\omega}} b$.*

2. *Let a, b be open λ-calculus typed terms. If $a \to_\beta b$, then the following hold:*

   (a) *there exists $c \in \vec{\Lambda s}_{op}$ such that $a \to_{\vec{\sigma}-gen} c \twoheadrightarrow_{\vec{s}_e} b$.*

(b) there exists $c \in \vec{\Lambda\omega}_{sop}$ *such that* $a \to_{\vec{\sigma}-gen} c \twoheadrightarrow_{\vec{\omega}_e} b$.

PROOF: All items can be proved by induction on the position where the reduction takes place. □

6.7 Weak Normalization of $\lambda\vec{\omega}_e$ and $\lambda\vec{s}_e$

The main technical tool in the proof of weak normalization of $\lambda\vec{\omega}_e$ is a translation similar to the one given in (37) of typed terms into functions whose arguments are λ-terms (or lists of them) and whose results are λ-terms (or lists of them). Although the idea is the same, the translation has to be carefully adapted. Let $\mathtt{T}$ be a given type. In order to define this translation we associate every term variable $X_{E,A}$ with a variable of the λ-calculus that we denote

$$\hat{X}_{A_1\to\ldots\to A_n\to\mathtt{T}\to A},$$

where $E = A_1, \ldots, A_n$. We also associate every substitution variable $x_{E,F}$ with a list of classical variables denoted

$$\hat{x}_{A_1\to\ldots\to A_n\to\mathtt{T}\to B_1}; \ldots; \hat{x}_{A_1\to\ldots\to A_n\to\mathtt{T}\to B_m}, \hat{x}_{A_1\to\ldots\to A_n\to\mathtt{T}\to\mathtt{T}},$$

where $F = B_1, \ldots, B_m$. The scripts show the types of the associated classical variables.

The translation maps every $\lambda\vec{\omega}$-term u such that $A_1, \ldots, A_n \vdash u : A$ into a function $[\![u]\!]$ whose arguments are lists of $n+1$ terms $t_1; \ldots; t_{n+1}$ of respective types $A_1, \ldots, A_n$ and $\mathtt{T}$ and which returns a term of type A. The translation of a $\lambda\vec{\omega}$-substitution s such that $A_1, \ldots, A_n \vdash s \triangleright B_1, \ldots, B_m$ is a function $[\![s]\!]$ whose arguments are lists of $n+1$ terms $t_1; \ldots; t_{n+1}$ of respective types $A_1, \ldots, A_n$ and $\mathtt{T}$ and which returns a list of $m+1$ λ-terms of types $B_1, \ldots, B_m$ and $\mathtt{T}$, respectively.

Essentially the translation reduces the term to substitution normal form, suspending substitutions on variables. Then, roughly speaking, substitution steps map to vacuous beta-reductions and $\vec{\sigma}$ $-gen$ (i.e. Beta) steps on substitution normal forms map to non-empty λ-calculus β reductions, yielding the desired result by a simulation argument.

Definition 6.7.1 *The translation* $[\![\bullet]\!]$ *is given as follows, where* $\overline{t}$ *denotes the list of terms*

$t_1;\ldots;t_n;t_{n+1}$ *and* $E = A_1,\ldots,A_n$, $F = B_1,\ldots,B_m$ *and* $E \to A = A_1 \to \ldots \to A_n \to \mathtt{T} \to A$.

1. $[\![X_{E,A}]\!](\overline{t}) = \hat{X}_{E\to A}\, t_1 \ldots t_{n+1}$
2. $[\![x_{E,F}]\!](\overline{t}) = (\hat{x}_{E\to B_1}\, t_1 \ldots t_{n+1});\cdots;(\hat{x}_{E\to B_m}\, t_1 \ldots t_{n+1});(\hat{x}_{E\to \mathtt{T}}\, t_1 \ldots t_{n+1})$
3. $[\![\mathtt{k}_E]\!](\overline{t}) = t_k$ *where* $k \leq n$
4. $[\![\lambda A.u]\!](\overline{t}) = \lambda z.([\![u]\!](z;\overline{t}))$ *with* z *fresh of type* A
5. $[\![u\, v]\!](\overline{t}) = ([\![u]\!](\overline{t}))([\![v]\!](\overline{t}))$
6. $[\![u[s]_i]\!](\overline{t}) = [\![u]\!](t_1;\ldots;t_{i-1};[\![s]\!](t_i;\ldots;t_{n+1}))$
7. $[\![\uparrow^k_E]\!](\overline{t}) = t_{k+1};\ldots;t_{n+1}$ *where* $k \leq n+1$
8. $[\![u/]\!](\overline{t}) = ([\![u]\!](\overline{t}));\overline{t}$

In item (4), z fresh means $z \notin FV(\overline{t})$ and $z \neq \hat{X}$, $\hat{x}$ for every free variable X and x in u. If we assume that our countable set of variables is ordered then we may take z as the first variable satisfying the previous conditions.

In item (6), the arguments list $(t_1;\ldots;t_{i-1};[\![s]\!](t_i;\ldots;t_{n+1}))$ should be interpreted as the concatenation between the list $t_1;\ldots;t_{i-1}$ and the list which results from $[\![s]\!](t_i;\ldots;t_{n+1})$.

Lemma 6.7.2 *If* $z \neq \hat{X},\hat{x}$ *for every* X,x *in* u, *then* $([\![u]\!](\overline{t}))[a/z] \equiv_\alpha [\![u]\!](\overline{t}[a/z])$ *where by* $\overline{t}[a/z]$ *we mean the list* $t_1[a/z];\ldots;t_n[a/z]$ *if* $\overline{t} = t_1;\ldots;t_n$.

PROOF: By an easy induction on u. □

The next lemma is important, stating that $[\![]\!]$ is invariant under all the rules of the substitution calculus.

Lemma 6.7.3 *Let* $f, g \in \Lambda\vec{\omega}_{op}$, *if* $f \to_{\vec{\omega}_e} g$ *then* $[\![f]\!] = [\![g]\!]$.

PROOF: By induction on the structure of f. If the reduction is internal, use the induction hypothesis. We only give the case where $f = a[s]_i$, $g = a[s']_i$ and $s \to s'$, since the application and abstraction cases are analogous.

Let $\overline{t} = t_1;\ldots;t_{n+1}$ with the right length. Then $[\![a[s]_i]\!](\overline{t}) =$
$[\![a]\!](t_1;\ldots;t_{i-1};[\![s]\!](t_i;\ldots;t_{n+1})) = [\![a]\!](t_1;\ldots;t_{i-1};[\![s']\!](t_i;\ldots;t_{n+1}))$
(by the induction hypothesis)
$= [\![a[s']_i]\!](\overline{t})$.

If the reduction is at the root, then we must study each rule. In all cases, let $\overline{t} = t_1;\ldots;t_{n+1}$ with the right length.

For the $\vec{\sigma}$-/-*des* rule,

- if $k > j$, then $[\![k[a/]_j]\!](t_1;\ldots;t_{n+1}) =$
  $[\![k]\!](t_1;\ldots;t_{j-1};[\![a/]\!](t_j;\ldots;t_{n+1})) =$
  $[\![k]\!](t_1;\ldots;t_{j-1};[\![a]\!](t_j;\ldots;t_{n+1});t_j;\ldots;t_{n+1}) =$
  $t_{k-1} = [\![k-1]\!](t_1;\ldots;t_{n+1})$
- if $k = j$, then $[\![k[a/]_j]\!](t_1;\ldots;t_{n+1}) =$
  $[\![k]\!](t_1;\ldots;t_{j-1};[\![a/]\!](t_j;\ldots;t_{n+1})) =$
  $[\![k]\!](t_1;\ldots;t_{j-1};[\![a]\!](t_j;\ldots;t_{n+1});t_j;\ldots;t_{n+1}) =$
  $[\![a]\!](t_j;\ldots;t_{n+1}) =$
  $[\![a]\!]([\![\uparrow^{j-1}]\!](t_1;\ldots;t_{n+1})) =$
  $[\![a[\uparrow^{j-1}]_1]\!](t_1;\ldots;t_{n+1})$
- if $k < j$, then $[\![k[a/]_j]\!](t_1;\ldots;t_{n+1}) =$
  $[\![k]\!](t_1;\ldots;t_{j-1};[\![a/]\!](t_j;\ldots;t_{n+1})) =$
  $t_k = [\![k]\!](t_1;\ldots;t_{n+1})$.

For the $\vec{\sigma}$-λ-*tr* rule,

$[\![(\lambda A.a)[s]_j]\!](t_1;\ldots;t_{n+1}) =$
$[\![(\lambda A.a)]\!](t_1;\ldots;t_{j-1};[\![s]\!](t_j;\ldots;t_{n+1})) =$
$\lambda z.[\![a]\!](z;t_1;\ldots;t_{j-1};[\![s]\!](t_j;\ldots;t_{n+1})) =$
$\lambda z.[\![a[s]_{j+1}]\!](z;t_1;\ldots;t_{n+1}) =$
$[\![\lambda A.(a[s]_{j+1})]\!](\overline{t})$.

For the $\vec{\sigma}$-*app*-*tr* rule,

$[\![(ab)[s]_j]\!](t_1;\ldots;t_{n+1}) =$
$[\![(ab)]\!](t_1;\ldots;t_{j-1};[\![s]\!](t_j;\ldots;t_{n+1})) =$
$[\![a]\!](t_1;\ldots;t_{j-1};[\![s]\!](t_j;\ldots;t_{n+1}))[\![b]\!](t_1;\ldots;t_{j-1};[\![s]\!](t_j;\ldots;t_{n+1})) =$
$[\![a[s]_j]\!](t_1;\ldots;t_{n+1})[\![b[s]_j]\!](t_1;\ldots;t_{n+1}) =$
$[\![a[s]_j b[s]_j]\!](t_1;\ldots;t_{n+1})$.

For the $\vec{\sigma}$-$\uparrow$-*des* rule,

- if $k \geq j$, then $[\![k[\uparrow^i]_j]\!](t_1;\ldots;t_{n+1}) =$
  $[\![k]\!](t_1;\cdots:t_{j-1}[\![\uparrow^i]\!](t_j;\ldots;t_{n+1})) =$
  $[\![k]\!](t_1;\cdots:t_{j-1};t_{j+i};\ldots;t_{n+1}) =$
  $t_{j+i+k-1-j+1} = t_{k+i} = [\![k+i]\!](t_1;\ldots;t_{n+1})$
- if $k < j$, then $[\![k[\uparrow^i]_j]\!](t_1;\ldots;t_{n+1}) =$
  $[\![k]\!](t_1;\cdots:t_{j-1}[\![\uparrow^i]\!](t_j;\ldots;t_{n+1})) =$
  $t_k = [\![k]\!](t_1;\ldots;t_{n+1})$.

For the $\vec{\sigma}$-/-*tr* rule, let $k \leq j$, $f = a[b/]_k[s]_j$ and $g = a[s]_{j+1}[b[s]_{j-k+1}/]_k$.
Let $B = [\![b]\!](t_k; \ldots; t_{j-1}; [\![s]\!](t_j; \ldots; t_{n+1}))$. Then
$[\![a[b/]_k[s]_j]\!](\overline{t}) =$
$[\![a[b/]_k]\!](t_1; \ldots; t_{j-1}; [\![s]\!](t_j; \ldots; t_{n+1})) =$
$[\![a]\!](t_1; \ldots; t_{k-1}; B; t_k; \ldots; t_{j-1}; [\![s]\!](t_j; \ldots; t_{n+1})) =$
$[\![a[s]_{j+1}]\!](t_1; \ldots; t_{k-1}; B; t_k; \ldots; t_{n+1}) =$
$[\![a[s]_{j+1}]\!](t_1; \ldots; t_{k-1}; [\![b[s]_{j-k+1}]\!](t_k; \ldots; t_{n+1})) =$
$[\![a[s]_{j+1}[b[s]_{j-k+1}/]_k]\!](\overline{t})$.

For the /-$\uparrow$-*tr* rule,

- if $k + i \leq j$, then $[\![a[\uparrow^i]_k[b/]_j]\!](\overline{t}) =$
  $[\![a[\uparrow^i]_k]\!](t_1; \ldots; t_{j-1}; [\![b]\!](t_j; \ldots; t_{n+1}); t_j \ldots; t_{n+1}) =$
  $[\![a]\!](t_1; \ldots; t_{k-1}; [\![\uparrow^i]\!](t_k; \ldots; t_{j-1}; [\![b]\!](t_j \ldots; t_{n+1}); t_j \ldots; t_{n+1})) =$
  $[\![a]\!](t_1; \ldots; t_{k-1}; t_{k+i}; \ldots; t_{j-1}; [\![b]\!](t_j \ldots; t_{n+1}); t_j \ldots; t_{n+1}) =$
  $[\![a[b/]_{j-i}]\!](t_1; \ldots; t_{k-1}; t_{k+i}; \ldots; t_{n+1}) =$
  $[\![a[b/]_{j-i}]\!](t_1; \ldots; t_{k-1}; [\![\uparrow^i]\!](t_k; \ldots; t_{n+1})) =$
  $[\![a[b/]_{j-i}[\uparrow^i]_k]\!](\overline{t})$
- if $k \leq j < k + i$, then $[\![a[\uparrow^i]_k[b/]_j]\!](\overline{t}) =$
  $[\![a[\uparrow^i]_k]\!](t_1; \ldots; t_{j-1}; [\![b]\!](t_j; \ldots; t_{n+1}); t_j \ldots; t_{n+1}) =$
  $[\![a]\!](t_1; \ldots; t_{k-1}; [\![\uparrow^i]\!](t_k; \ldots; t_{j-1}; [\![b]\!](t_j \ldots; t_{n+1}); t_j \ldots; t_{n+1})) =$
  $[\![a]\!](t_1; \ldots; t_{k-1}; t_{k+i-1}; \ldots; t_{n+1}) =$
  $[\![a]\!](t_1; \ldots; t_{k-1}; [\![\uparrow^{i-1}]\!](t_k; \ldots; t_{n+1}); t_k; \ldots; t_{n+1}) =$
  $[\![a[\uparrow^{i-1}]_k]\!](\overline{t})$.

For the $\uparrow$-$\uparrow$-*tr* rule,

- if $k + i < j$, then $[\![a[\uparrow^i]_k[\uparrow^l]_j]\!](\overline{t}) =$
  $[\![a[\uparrow^i]_k]\!](t_1; \ldots; t_{j-1}; [\![\uparrow^l]\!](t_j; \ldots; t_{n+1})) =$
  $[\![a[\uparrow^i]_k]\!](t_1; \ldots; t_{j-1}; t_{j+l}; \ldots; t_{n+1}) =$
  $[\![a]\!](t_1; \ldots; t_{k-1}; [\![\uparrow^i]\!](t_k; \ldots; t_{j-1}; t_{j+l}; \ldots; t_{n+1})) =$
  $[\![a]\!](t_1; \ldots; t_{k-1}; t_{k+i}; \ldots; t_{j-1}; t_{j+l}; \ldots; t_{n+1}) =$
  $[\![a]\!](t_1; \ldots; t_{k-1}; t_{k+i}; \ldots; t_{j-i-1+i}; [\![\uparrow^l]\!](t_k; \ldots; t_{n+1})) =$
  $[\![a[\uparrow^l]_{j-i}]\!](t_1; \ldots; t_{k-1}; [\![\uparrow^i]\!](t_k; \ldots; t_{n+1})) =$
  $[\![a[\uparrow^l]_{j-i}[\uparrow^i]_k]\!](\overline{t})$
- if $k \leq j \leq k + i$, then $[\![a[\uparrow^i]_k[\uparrow^l]_j]\!](\overline{t}) =$
  $[\![a[\uparrow^i]_k]\!](t_1; \ldots; t_{j-1}; [\![\uparrow^l]\!](t_j; \ldots; t_{n+1})) =$
  $[\![a]\!](t_1; \ldots; t_{k-1}; [\![\uparrow^i]\!](t_k; \ldots; t_{j-1}; t_{j+l}; \ldots; t_{n+1})) =$

$[\![a]\!](t_1;\ldots;t_{k-1};t_{k+i+l};\ldots;t_{n+1}) =$
$[\![a]\!](t_1;\ldots;t_{k-1};[\![\uparrow^{i+l}]\!](t_k;\ldots;t_{n+1})) =$
$[\![a[\uparrow^{i+l}]_k]\!](\overline{t})$

□

Definition 6.7.4 *Given the terms $u, v \in \Lambda\vec{\omega}_{op}$ we say that u and v* have the same type *if both are well typed and there is a context E and type A such that $E \vdash_{\mathbf{Ls\vec{1}}} u : A$ and $E \vdash_{\mathbf{Ls\vec{1}}} v : A$.*

Definition 6.7.5 *We define the quasi-order $>$ on $\Lambda\vec{\omega}_{op}$ terms as follows: $u > v$ if u and v have the same type and $[\![u]\!](\overline{t}) \xrightarrow{+}_{\beta} [\![v]\!](\overline{t})$ for every list of λ-terms $\overline{t}$ of the right length and the right types (to be called* right *$\overline{t}$ from now onwards).*

It follows immediately that $>$ is a strict order (i.e. irreflexive and transitive), which is also compatible with taking closures as stated next.

Lemma 6.7.6 *1. Let $a, b \in \Lambda\,\vec{\omega}^{t}_{sop}$, $j \geq 1$ and $s \in \Lambda\,\vec{\omega}^{s}_{sop}$. If $a > b$, then $a[s]_j > b[s]_j$.*

2. Let $a, b \in \Lambda\,\vec{\omega}^{t}_{sop}$, $i_1, \ldots, i_k \geq 1$ and $s_1, \ldots, s_k \in \Lambda\,\vec{\omega}^{s}_{sop}$. If $a > b$, then $a[s_1]_{i_1} \ldots [s_k]_{i_k} > b[s_1]_{i_1} \ldots [s_k]_{i_k}$.

Proof:

1. For every right $\overline{t}$,
   $[\![a[s]_j]\!](\overline{t}) = [\![a]\!](t_1, \ldots, t_{j-1}, [\![s]\!](t_j, \ldots, t_n)) \rightarrow^{+}_{\beta} [\![b]\!](t_1, \ldots, t_{j-1}, [\![s]\!](t_j, \ldots, t_n))$
   $= [\![b[s]_j]\!](\overline{t})$.
2. Iterating the previous result.

□

The following Lemma is technically important and will be used in the proof of Lemma 6.7.8.

Lemma 6.7.7 *Let $k \geq 1$, $i_1 \geq i_2 \geq \cdots \geq i_k \geq 1$, $s_1, \ldots, s_k \in \Lambda\,\vec{\omega}^{s}_{sop}$ and let X be a term variable. Then for every right $\overline{t}$, there exist $r \geq 1, q_1, \ldots, q_r \in \Lambda\,\vec{\omega}^{t}_{sop}$ which do not depend on $t_1, \ldots, t_{i_k-1}$ such that $[\![X[s_1]_{i_1} \ldots [s_k]_{i_k}]\!](\overline{t}) = \hat{X}\ t_1\ t_2\ \ldots\ t_{i_k-1}\ q_1\ \ldots\ q_r$.*

Proof: By induction on k.

- For $k = 1$ we have:
  - $s_1 = d/$, then $[\![X[s_1]_{i_1}]\!](\overline{t}) = \hat{X}\ t_1\ \ldots\ t_{i_1-1}\ [\![d]\!](t_{i_1}, \ldots, t_n)\ t_{i_1}\ \ldots\ t_n$, thus take $(q_1, \ldots, q_r) = ([\![d]\!](t_{i_1}, \ldots, t_n), t_{i_1}, \ldots, t_n)$ which clearly do not depend on the prescribed terms.

– $s_1 = \uparrow^m$ with $m \geq 0$, then $[\![X[s_1]_{i_1}]\!](\overline{t}) = \hat{X}\ t_1\ \ldots\ t_{i_1-1}\ t_{i_1+m}\ \ldots\ t_n$, thus take $(q_1, \ldots, q_r) = (t_{i_1+m}, \ldots, t_n)$ which do not depend on the prescribed terms either.

- For the inductive case we have:
  – $s_{k+1} = d/$, then $[\![X[s_1]_{i_1} \ldots [s_k]_{i_k}[s_{k+1}]_{i_{k+1}}]\!](\overline{t})$
  $= [\![X[s_1]_{i_1} \ldots [s_k]_{i_k}]\!](t_1, \ldots, t_{i_{k+1}-1}[\![d]\!](t_{i_{k+1}}, \ldots, t_n), t_{i_{k+1}}, \ldots, t_n)$
  $= \hat{X}\ t_1\ \ldots\ t_{i_{k+1}-1}\ [\![d]\!](t_{i_{k+1}}, \ldots, t_n)\ t_{i_{k+1}}\ \ldots\ t_{i_k-2}\ q_1\ \ldots\ q_r$ (by the induction hypothesis) where $q_1, \ldots, q_r$ do not depend on the previous terms in the list.
  Taking $(q'_1, \ldots, q'_{r'}) = ([\![d]\!](t_{i_{k+1}}, \ldots, t_n), t_{i_{k+1}}, \ldots, t_{i_k-2}, q_1, \ldots, q_r)$,
  no q'_j depends on $t_1, \ldots, t_{i_{k+1}-1}$, and
  $[\![X[s_1]_{i_1} \ldots [s_{k+1}]_{i_{k+1}}]\!](\overline{t}) = \hat{X}\ t_1\ t_2\ \ldots\ t_{i_{k+1}-1}\ q'_1\ \ldots\ q'_{r'}$.
  – $s_{k+1} = \uparrow^m$ with $m \geq 0$, then $[\![X[s_1]_{i_1} \ldots [s_k]_{i_k}[s_{k+1}]_{i_{k+1}}]\!](\overline{t})$
  $= [\![X[s_1]_{i_1} \ldots [s_k]_{i_k}]\!](t_1, \ldots, t_{i_{k+1}-1}, t_{i_{k+1}+m}, \ldots, t_n)$
  $= \hat{X}\ t_1\ \ldots\ t_{i_{k+1}-1}\ t_{i_{k+1}+m}\ \ldots\ t_{i_k-1+m}\ q_1\ \ldots\ q_r$ (by the induction hypothesis) where $q_1, \ldots, q_r$ do not depend on the previous terms in the list. Taking $(q'_1, \ldots, q'_{r'}) = (t_{i_{k+1}+m}, \ldots, t_{i_k-1+m}, q_1, \ldots, q_r)$, no q'_j depends on $t_1, \ldots, t_{i_{k+1}-1}$, and
  $[\![X[s_1]_{i_1} \ldots [s_{k+1}]_{i_{k+1}}]\!](\overline{t}) = \hat{X}\ t_1\ t_2\ \ldots\ t_{i_{k+1}-1}\ q'_1\ \ldots\ q'_{r'}$.

□

Now we can give the key result, for semi-open terms.

Lemma 6.7.8 *Let $a, b \in \Lambda\ \overrightarrow{\omega}^t_{sop}$ where a is an $\overrightarrow{\omega}_e$-normal form. If $a \to_{\overrightarrow{\sigma}-gen} b$ then $a > b$.*

PROOF: By induction on the position of the $\overrightarrow{\sigma}$ $-gen$ redex in the term a.

If the reduction is at the root, i.e. $a = (\lambda A.c)d$ and $b = c[d/]_1$, then
$[\![(\lambda A.c)d]\!](\overline{t}) = ([\![(\lambda A.c)]\!](\overline{t}))([\![d]\!](\overline{t})) = (\lambda z.[\![c]\!](z;\overline{t}))([\![d]\!](\overline{t})) \to_\beta [\![c]\!](z;\overline{t}))[([\![d]\!](\overline{t}))/z]$
$= [\![c]\!]([\![d]\!](\overline{t});\overline{t}) = [\![c[d/]_1]\!](\overline{t})$ (by Lemma 6.7.2). Remark that, when Lemma 6.7.2 has been used, since z should be chosen such that $z \notin FV(\overline{t})$, then $\overline{t}[([\![d]\!](\overline{t}))/z] = \overline{t}$.

Else we have the following cases:

- $a = n$ or $a = X$ a term variable, the result holds vacuously since there is no $\overrightarrow{\sigma}$ $-gen$ redex.
- $a = cd$ then if the reduction occurs in c, say $c \to_{\overrightarrow{\sigma}-gen} c'$, we have
  $[\![a]\!](\overline{t}) = [\![c]\!](\overline{t})[\![d]\!](\overline{t}) \to^+_\beta [\![c']\!](\overline{t})[\![d]\!](\overline{t}) = [\![(c'd)]\!](\overline{t}) = [\![b]\!](\overline{t})$ using the induction hypothesis; and the situation is analogous if the reduction occurs in d.
- $a = \lambda A.c$ then the reduction occurs in c, say $c \to_{\overrightarrow{\sigma}-gen} c'$, then
  $[\![a]\!](\overline{t}) = \lambda z.[\![c]\!](\overline{t}) \to^+_\beta \lambda z.[\![c']\!](\overline{t}) = [\![\lambda A.c']\!](\overline{t}) = [\![b]\!](\overline{t})$ using the induction hypothesis.

- a is a closure, then a will necessarily have the form $d[s_1]_{i_1} \dots [s_m]_{i_m}$ where d is not a closure; d cannot be an abstraction, nor an application, nor an index (or else a would not be a nf). Then $d = X$ a term variable, and $i_1 > i_2 > \dots > i_m$ (or else a would not be a nf). Then we have that there exists $k \geq 1$ such that $a = X[s_1]_{i_1} \dots [s_{k-1}]_{i_{k-1}}[s_k]_{i_k}[s_{k+1}]_{i_{k+1}} \dots [s_m]_{i_m}$ where $s_k = e/$ with $e \to_{\vec{\sigma}-gen} e'$, for some terms e, e', and
$a \to_{\vec{\sigma}-gen} X[s_1]_{i_1} \dots [s_{k-1}]_{i_{k-1}}[e'/]_{i_k}[s_{k+1}]_{i_{k+1}} \dots [s_m]_{i_m} = b$.

  Suppose first $k \geq 2$. In what follows Lemma 6.7.7 will be used twice; the non dependence of the terms $q_1, \dots, q_r$ on the $i_{k-1} - 1$ terms explicited in the proof guarantees that the $q_1, \dots, q_r$ which appear after the 2nd. equality also ensure that the 3rd. equality holds. We have that
  $[\![X[s_1]_{i_1} \dots [s_{k-1}]_{i_{k-1}}[e/]_{i_k}]\!](\overline{t})$
  $= [\![X[s_1]_{i_1} \dots [s_{k-1}]_{i_{k-1}}]\!](t_1, \dots, t_{i_k-1}, [\![e]\!](t_{i_k}, \dots, t_n), t_{i_k}, \dots, t_n)$
  $= \hat{X}\ t_1\ t_2\ \dots\ t_{i_k-1}\ [\![e]\!](t_{i_k} \dots t_n)\ t_{i_k}\ \dots\ t_{i_{k-1}-1}\ q_1\ \dots\ q_r$ (by Lemma 6.7.7)
  $\to_\beta^+ \hat{X}\ t_1\ t_2\ \dots\ t_{i_k-1}\ [\![e']\!](t_{i_k} \dots t_n)\ t_{i_k}\ \dots\ t_{i_{k-1}-1}\ q_1\ \dots\ q_r$
  (by the induction hypothesis and compatibility).

  On the other hand,
  $[\![X[s_1]_{i_1} \dots [s_{k-1}]_{i_{k-1}}[e'/]_{i_k}]\!](\overline{t})$
  $= [\![X[s_1]_{i_1} \dots [s_{k-1}]_{i_{k-1}}]\!](t_1, \dots, t_{i_k-1}, [\![e']\!](t_{i_k}, \dots, t_n), t_{i_k}, \dots, t_n)$
  $= \hat{X}\ t_1\ t_2\ \dots\ t_{i_k-1}\ [\![e']\!](t_{i_k} \dots t_n)\ t_{i_k}\ \dots\ t_{i_{k-1}-1}\ q'_1\ \dots\ q'_r$
  and because of the fact that $q'_1, \dots, q'_r$ do not depend on the terms before q'_1, we have that $q_i = q'_i$ for $1 \leq i \leq r$.

  Thus, by Lemma 6.7.6(2),
  $[\![a]\!](\overline{t}) = [\![X[s_1]_{i_1} \dots [s_{k-1}]_{i_{k-1}}[e/]_{i_k}[s_{k+1}]_{i_{k+1}} \dots [s_m]_{i_m}]\!](\overline{t})$
  $\to_\beta^+ [\![X[s_1]_{i_1} \dots [s_{k-1}]_{i_{k-1}}[e'/]_{i_k}[s_{k+1}]_{i_{k+1}} \dots [s_m]_{i_m}]\!](\overline{t}) = [\![b]\!](\overline{t})$.

  The case $k = 1$ follows directly by Definition 6.7.1:
  $[\![X[e/]_{i_1}]\!](\overline{t})$
  $= \hat{X}\ t_1\ t_2\ \dots\ t_{i_1-1}\ [\![e]\!](t_{i_1} \dots t_n)\ t_{i_1}\ \dots\ t_n$
  $\to_\beta^+ \hat{X}\ t_1\ t_2\ \dots\ t_{i_1-1}\ [\![e']\!](t_{i_1} \dots t_n)\ t_{i_1}\ \dots\ t_n$
  (by the induction hypothesis and compatibility)
  $= [\![X[e'/]_{i_1}]\!](\overline{t})$ and use Lemma 6.7.6(2) similarly.

□

Even though it is an adaptation of the technique in (37), the technical result in Lemma 6.7.7 was required. In Lemma 6.7.8 special care is necessary for handling closures. For the case that a is a closure, if it were any open term (i.e. having substitution variables), it might have a more complicated form than just $d[s_1]_{i_1} \dots [s_m]_{i_m}$ with the conditions above mentioned, and here is where the method would not work. The condition $i_1 > \dots > i_m$ becomes strictly necessary, otherwise the β-redex could be lost in the $[\![\bullet]\!]$ transform. The result does not hold for full open terms, taking for instance the normal form $a = 2[x]_2[d/]_1$ as a counterexample, for $d \to_{\vec{\sigma}-gen} d'$, since $a \to_{\vec{\sigma}-gen} 2[x]_2[d'/]_1$ but
$[\![a]\!](\bar{t}) = [\![2[x]_2]\!]([\![d/]\!](t_1; \dots; t_n)) =$
$[\![2[x]_2]\!]([\![d]\!](t_1; \dots; t_n); t_1; \dots; t_n) =$
$[\![2]\!]([\![d]\!](t_1; \dots; t_n); [\![x]\!](t_1; \dots; t_n)) =$
$[\![2]\!]([\![d]\!](t_1; \dots; t_n); \hat{x}t_1 \dots t_n; \dots; \hat{x}t_1 \dots t_n) =$
$\hat{x}t_1 \dots t_n$
which may not have β-redexes. This counterexample is critical. So far, Lemma 6.7.8 could not be extended for the full open term set using the same technique; it does not seem that some simple or intuitive change in the $[\![\bullet]\!]$ function definition could help. Problems also are caused by terms like $1[x]_2$, which could be partially fixed redefining the $\vec{\lambda\omega_e}$-calculus in such a way that terms of the form $k[x]_j$ for $k < j$ would not be in normal form. Recall that if $a, b \in \vec{\Lambda\omega}$ are two typed terms where a is an $\vec{\omega_e}$-normal form, we need to certify that, when $a \to_{\vec{\sigma}-gen} b$, then for every right $\bar{t}$, $[\![a]\!](\bar{t}) \xrightarrow{+}_{\beta} [\![b]\!](\bar{t})$, i.e. the possibility of β reduction in the simply typed λ-calculus must be created. For this reason we could add to $\vec{\lambda\omega_e}$ the following rule:

$$(\vec{\sigma}\ -des) \quad n[s]_j \quad \to \quad n \quad if\ n < j$$

This rule subsumes rules ($\vec{\sigma}$-/-des) and ($\vec{\sigma}$-$\uparrow$-des) for the case $n < j$, and it is consistent with Lemma 6.7.3. The addition of this rule forces terms like $n[x]_j$ for $n < j$ not to be $\vec{\omega_e}$-normal forms. But note that terms having closures with substitution variables are not in the domain of the translation S.

Nevertheless, the addition of this new rule does not fix the problem for examples like a above, since the condition is $n = j$ and there is no similar rule to be added in order to force $n[x]_n$ to reduce.

In virtue of the main result in (37), a natural question is: what happens in $\lambda\omega_e$ which can make the difference with $\lambda\sigma$, since the latter is WN on all typed open terms? The reason we found is that, when the counterexample is translated to $\lambda\sigma$, it is not a σ-normal form and hence it does not represent a counterexample. Remark that the statement of Lemma 6.7.8 is for $\vec{\omega_e}$-normal forms. More precisely, $a = 2[x]_2[d/]_1$ translates to the $\lambda\sigma$ term $1[\uparrow][1.(x \circ \uparrow)][d.id]$ (and to the $\lambda\sigma_{\Uparrow}$ term $1[\uparrow][\Uparrow (x)][d.id]$), which is clearly not a σ-normal form.

The problem forced the statement of Lemma 6.7.8 to refer to semi-open terms. It is worth mentioning, however, that due to the above isomorphism it suffices to consider this restricted set of terms in order to obtain WN of $\vec{\lambda s}_e$. This was our original goal.

Definition 6.7.9 *A* canonical strategy *for $\lambda\vec{\omega}_e$ is a strategy which applies the $\vec{\sigma}$ $-gen$ rule only to $\vec{\sigma}$ $-gen$-redexes in $\vec{\omega}_e$-normal forms and whose $\vec{\omega}_e$-reductions are normalizing.*

As an example, take a strategy which applies the $\vec{\sigma}$ $-gen$ rule only to $\vec{\omega}_e$-normal forms and whose $\vec{\omega}_e$-reductions are leftmost-innermost (li). Hence, given a term a_1, such a canonical reduction sequence will be:

$$a_1 \overset{li}{\twoheadrightarrow}_{\vec{\omega}_e} \vec{\omega}_e\ (a_1) \rightarrow_{\vec{\sigma}-gen} a_2 \overset{li}{\twoheadrightarrow}_{\vec{\omega}_e} \vec{\omega}_e\ (a_2) \rightarrow_{\vec{\sigma}-gen} \cdots$$

where $\overset{li}{\twoheadrightarrow}_{\vec{\omega}_e}$ stands for li $\vec{\omega}_e$-reduction and $\vec{\omega}_e\ (a_i)$ is the $\vec{\omega}_e$-normal form of a_i.

Theorem 6.7.10 *Every canonical strategy for $\lambda\vec{\omega}_e$ is strongly normalizing and therefore the $\lambda\vec{\omega}_e$-calculus is WN for semi-open terms.*

PROOF: If there is an infinite reduction sequence

$$a_1 \twoheadrightarrow_{\vec{\omega}_e} \vec{\omega}_e\ (a_1) \rightarrow_{\vec{\sigma}-gen} a_2 \twoheadrightarrow_{\vec{\omega}_e} \vec{\omega}_e\ (a_2) \rightarrow_{\vec{\sigma}-gen} \cdots$$

then by Lemmas 6.7.3 and 6.7.8, for every right $\overline{t}$, we get a contradiction through the infinite reduction sequence in the typed λ-calculus:

$$[\![a_1]\!](\overline{t}) = [\![\vec{\omega}_e\ (a_1)]\!](\overline{t}) \rightarrow_\beta^+ [\![a_2]\!](\overline{t}) = [\![\vec{\omega}_e\ (a_2)]\!](\overline{t}) \rightarrow_\beta^+ \cdots$$

□

Now, the isomorphism presented in Section 6.3, gives:

Theorem 6.7.11 *The $\vec{\lambda s}_e$-calculus is weakly normalizing for open terms.*

6.8 The $\lambda\omega'_e$-calculus.

We extend the previous result to a new calculus, $\vec{\lambda\omega'_e}$, derived from $\vec{\lambda\omega}_e$. In this section we omit typing decorations for notation simplicity, therefore $\vec{\lambda\omega'_e}$ will be just written $\lambda\omega'_e$.

$\lambda\omega'_e$ is written in the style of $\lambda\sigma$ and has 1 as the sole de Bruijn index, while the others are constructed as in $\lambda\sigma$. We will show that typed $\lambda\omega'_e$ is WN on semi-open terms.

A good reason to use this calculus is to show the power of the composition rules, which indeed emulate the behavior of the other indices. Thus with a smaller language one will have in some sense the same reduction possibilities.

Remark that the problem which forced us to restrict Lemma 6.7.8 still holds. Up to now, we do not know whether typed $\lambda\omega'_e$ is WN on all open terms.

$$
\begin{array}{llcll}
(\sigma\text{-}gen') & (\lambda a)b & \longrightarrow & a[b/]_1 & \\
(\sigma\text{-}app\text{-}tr') & (ab)[s]_j & \longrightarrow & a[s]_j b[s]_j & \\
(\sigma\text{-}\lambda\text{-}tr') & (\lambda a)[s]_j & \longrightarrow & \lambda(a[s]_{j+1}) & \\
(\sigma\text{-}/\text{-}des') & 1[a/]_1 & \longrightarrow & a[\uparrow^0]_1 & \\
(\sigma\text{-}\uparrow\text{-}des') & 1[\uparrow^0]_1 & \longrightarrow & 1 & \\
(\sigma\text{-}des') & 1[s]_j & \longrightarrow & 1 & j > 1 \\
(\sigma\text{-}/\text{-}tr') & a[b/]_k[s]_j & \longrightarrow & a[s]_{j+1}[b[s]_{j-k+1}/]_k & k \leq j \\
(/\text{-}\uparrow\text{-}tr') & a[\uparrow^i]_k[b/]_j & \longrightarrow & \begin{cases} a[b/]_{j-i}[\uparrow^i]_k & k+i \leq j \\ a[\uparrow^{i-1}]_k & k \leq j < k+i \end{cases} & \\
(\uparrow\text{-}\uparrow\text{-}tr') & a[\uparrow^i]_k[\uparrow^l]_j & \longrightarrow & \begin{cases} a[\uparrow^l]_{j-i}[\uparrow^i]_k & k+i < j \\ a[\uparrow^{i+l}]_k & k \leq j \leq k+i \end{cases} &
\end{array}
$$

Figure 6.7: The rewriting rules of the simply typed $\lambda\omega'_e$-calculus

Definition 6.8.1 *The set of open terms and substitutions of the $\lambda\omega'_e$-calculus, noted $\Lambda\omega'_{op}$, is defined as $\Lambda^t\omega'_{op} \cup \Lambda^s\omega'_{op}$, where $\Lambda^t\omega'_{op}$ (terms) and $\Lambda^s\omega'_{op}$ (substitutions) are mutually defined as follows:*

$$
\begin{array}{llcll}
\textbf{Open terms} & a & ::= & X \mid 1 \mid (\lambda\ a) \mid (a\ a) \mid a[s]_j & j \geq 1 \\
\textbf{Open substitutions} & s & ::= & x \mid \uparrow^k \mid a\ / & k \geq 0
\end{array}
$$

and the set of semi-open terms and substitutions of the $\lambda\omega'_e$-calculus is defined as $\Lambda^t\omega'_{sop} \cup \Lambda^s\omega'_{sop}$, where $\Lambda^t\omega'_{sop}$ (terms) and $\Lambda^s\omega'_{sop}$ (substitutions) are mutually defined as follows:

$$
\begin{array}{llcll}
\textbf{Semi-open terms} & a & ::= & X \mid 1 \mid (\lambda\ a) \mid (a\ a) \mid a[s]_j & j \geq 1 \\
\textbf{Semi-open substitutions} & s & ::= & \uparrow^k \mid a\ / & k \geq 0
\end{array}
$$

where X ranges over $\mathbf{V}$ a denumerably infinite set of term variables and x ranges over $\mathbf{W}$ a denumerably infinite set of substitution variables. The rules of the $\lambda\omega'_e$-calculus are given in Figure 6.7

As with $\lambda\omega_e$, all rules except *(σ-gen')* conform ω'_e.

Note that the *(σ-des')* rule, in the presence of the *(σ-/-des')* rule, subsumes the following two possible rules:

$$
\begin{array}{llcll}
(\sigma - / - des'') & 1[a/]_j & \rightarrow & \begin{cases} a[\uparrow^0]_1 & j = 1 \\ 1 & j > 1 \end{cases} \\
(\sigma - \uparrow - des'') & 1[\uparrow^i]_j & \rightarrow & 1 \quad j > 1
\end{array}
$$

Note also that for all $i \geq 1$, the term $1[\uparrow^i]_1$ is an ω'_e-normal form representing the de Bruijn index $i+1$. In fact, $1[\uparrow^i]_1[\uparrow^l]_1 \to_{\omega'_e} 1[\uparrow^{i+l}]_1$.

Now we wish to relate $\lambda\omega'_e$ and $\lambda\omega_e$ by means of a translation.

Definition 6.8.2 *For open terms and substitutions we define a translation* $|\bullet| : \Lambda\omega_{op} \to \Lambda\omega'_{op}$ *by:*

$$\begin{array}{lcllcl} |X| & = & X & |x| & = & x \\ |1| & = & 1 & |n+1| & = & 1[\uparrow^n]_1 \quad (n \geq 1) \\ |\lambda a| & = & \lambda|a| & |ab| & = & |a||b| \\ |\uparrow^k| & = & \uparrow^k & |a/| & = & |a|/ \\ |a[s]_j| & = & |a|[|s|]_j & & & \end{array}$$

Note that the translation of an index greater than 1 yields a term of the form $1[\uparrow^n]$, while 1 is translated as 1.

We give a Simulation Proposition which will be used in the subsequent results of the section.

Proposition 6.8.3 (Simulation) *Let* $a, b \in \Lambda\omega_{op}$.

1. *If* $a \to_{\sigma-gen} b$, *then* $|a| \to_{\sigma-gen'} |b|$.
2. *If* $a \to_{\omega_e} b$, *then* $|a| \twoheadrightarrow_{\omega'_e} |b|$.
3. *If* $a \twoheadrightarrow_{\omega_e} b$, *then* $|a| \twoheadrightarrow_{\omega'_e} |b|$.
4. *If* $a \twoheadrightarrow_{\lambda\omega_e} b$, *then* $|a| \twoheadrightarrow_{\lambda\omega'_e} |b|$.

PROOF:

1. By induction on a. If the reduction is at the root where $(\lambda c)d \to_{\sigma-gen} c[d/]_1$, then $|a| = (\lambda|c|)|d| \to_{\sigma-gen'} |c|[|d|/]_1 = |b|$. For internal reductions, the proof is straightforward.
2. By induction on a. If the reduction is at the root, we analyze every possible ω_e-rule applied.
   - $a = n[c/]_j \to_{\sigma-/-des} n-1 = b$ with $n > j \geq 1$, then $n \geq 2$. If $n > 2$, then $|a| = |n|[|c|/]_j = 1[\uparrow^{n-1}]_1[|c|/]_j \to_{/-\uparrow-tr'} 1[\uparrow^{n-2}]_1 = |n-1| = |b|$, since $1 = k \leq j < k+i = n > 2$. If $n = 2$, $1[\uparrow^{n-2}]_1 = 1[\uparrow^0]_1 \to_{\sigma-\uparrow-des'} 1 = |1| = |b|$.
   - $a = n[c/]_j \to_{\sigma-/-des} c[\uparrow^{j-1}]_1 = b$ with $n = j \geq 1$.
     If $n = 1$, then $|n[c/]_j| = 1[|c|/]_j \to_{\sigma-/-des'} |c|[\uparrow^0]_1 = |c[\uparrow^0]_1|$ and we are done.
     If $n > 1$, then $|a| = |n|[|c|/]_j = 1[\uparrow^{n-1}]_1[|c|/]_j \to_{/-\uparrow-tr'} 1[|c|/]_{j-n+1}[\uparrow^{n-1}]_1$
     $\to_{\sigma-\uparrow-des'} |c|[\uparrow^0]_1[\uparrow^{n-1}]_1 \to_{\uparrow-\uparrow-tr'} |c|[\uparrow^{n-1}]_1 = |b|$ since
     $1 = k \leq k+i = n = j$ thus $j-n+1 = 1$.

- $a = n[c/]_j \to_{\sigma-/-des} n = b$ with $1 \leq n < j$. If $n = 1$, we are done by rule (σ-/-des') since $j > 1$. Else, $|a| = |n|[|c|/]_j = 1[\uparrow^{n-1}]_1[|c|/]_j \to_{/-\uparrow-tr'} 1[|c|/]_{j-n+1}[\uparrow^{n-1}]_1 \to_{\sigma-des'} 1[\uparrow^{n-1}]_1 = |b|$, since $j > n = k + i \leq j$ thus $j - n + 1 > 1$.
- $a = n[\uparrow^l]_j \to_{\sigma-\uparrow-des} n + l = b$, with $n \geq j \geq 1$. We have the following cases
  If $n = 1$ (thus $j = 1$) and $l = 0$, then $|a| = 1[\uparrow^0]_1 \to_{\sigma-\uparrow-des'} 1 = |b|$.
  If $n = 1$ (thus $j = 1$) and $l > 0$, then $|a| = 1[\uparrow^l]_1 = |1 + l| = |b|$
  If $n \geq 2$ then $|a| = |n|[\uparrow^l]_j = 1[\uparrow^{n-1}]_1[\uparrow^l]_j \to_{\uparrow-\uparrow-tr'} 1[\uparrow^{n+l-1}]_1 = |n + l| = |b|$
  since $1 = k \leq j \leq k + i = n$.
- $a = n[\uparrow^l]_j \to_{\sigma-\uparrow-des} n = b$ with $1 \leq n < j$. We have the following cases:
  If $n = 1$, then $|a| = 1[\uparrow^l]_j \to_{\sigma-des'} 1 = |b|$ since $j > 1$.
  If $n \geq 2$ then $|a| = 1[\uparrow^{n-1}]_1[\uparrow^l]_j \to_{\uparrow-\uparrow-tr'} 1[\uparrow^l]_{j-n+1}[\uparrow^{n-1}]_1 \to_{\sigma-des'} 1[\uparrow^{n-1}]_1$
  $= |n| = |b|$ since $1 \leq n < j$ thus $j - n + 1 > 1$.
- The other rules are straightforward.

For internal reductions, the proof is straightforward.

3. Consequence of the second item.
4. Consequence of the previous items.

□

As it can be seen in the proof of Proposition 6.8.3, the ω'_e-rules (σ-/-tr'), (/-↑-tr') and (↑-↑-tr') can handle closures over indices thus simulating the behavior of the ω_e-rules (σ-/-des) and (σ-↑-des).

Remark 6.8.4 *Let $u, v \in \Lambda\omega'_{sop}$.*

1. *If $u \to_{\lambda\omega'_e} v$, then $u \to_{\lambda\omega_e} v$.*
2. *$|u| = u$.*

Proof: Both 1. and 2. can be proved by an easy induction on u. □

The second assertion above means that the translation is onto and invariant for the set $\Lambda\omega'_{sop}$.

Corollary 6.8.5 (Confluence) *$\lambda\omega'_e$ and ω'_e are confluent on semi-open terms.*

Proof: To prove the confluence of $\lambda\omega'_e$, let $a \in \Lambda\omega'_{sop}$, and suppose $a \twoheadrightarrow_{\lambda\omega'_e} a_1$, $a \twoheadrightarrow_{\lambda\omega'_e} a_2$. By Remark 6.8.4, both derivations are also $\lambda\omega_e$-derivations. Since $\lambda\omega_e$ is isomorphic to λs_e, it is confluent on semi-open terms (46), thus there exists $b \in \Lambda\omega_{sop}$ such that $a_1 \twoheadrightarrow_{\lambda\omega_e} b$ and $a_2 \twoheadrightarrow_{\lambda\omega_e} b$. By the fourth item of the Simulation Proposition, $|a_1| \twoheadrightarrow_{\lambda\omega'_e} |b|$ and $|a_2| \twoheadrightarrow_{\lambda\omega'_e} |b|$. Since $a = |a|$ by Remark 6.8.4, this closes the diagram.

The confluence of ω'_e is proved analogously, by using the third item of the Simulation Proposition. □

We define the typing rules of $\lambda\omega'_e$ in a straightforward manner analogously to $\vec{\lambda\omega}_e$. Moreover, we have:

Lemma 6.8.6 (Typability preservation) *For all $a \in \Lambda\omega'_{sop}$, if a is typed in $\lambda\omega'_e$, then a is typed in $\lambda\omega_e$.*

PROOF: By induction on a. □

6.8.1 Weak normalization of typed $\lambda\omega'_e$

In order to prove WN of typed $\lambda\omega'_e$, we will relate the $\lambda\omega_e$-calculus with the $\lambda\omega'_e$-calculus. We first give a grammar for the set of $\lambda\omega_e$ open terms in ω_e-normal form and another grammar for the set of $\lambda\omega_e$ open terms in $\lambda\omega_e$-normal form. These grammars will specify conditions associated to some of their rules (strictly speaking, they can be seen as grammar schemas or conditional grammars.)

We denote with NF_{ω_e}, $NF_{\lambda\omega_e}$, $NF_{\omega'_e}$ and $NF_{\lambda\omega'_e}$ the sets of normal forms of the respective calculi untyped open terms.

Definition 6.8.7 *We call ω_e-syntactic normal forms the terms NS_{ω_e} generated by the following syntax with start symbol M:*

$$
\begin{array}{lcll}
M & ::= & M_1 \dots M_n \mid c \mid \lambda M & \text{where } n \geq 1 \\
c & ::= & c_1 \mid c_2 & \\
c_1 & ::= & m[s_1]_{i_1} \dots [s_n]_{i_n} & \text{where } m \geq 1, n \geq 0, \forall 1 \leq k \leq n, i_k \geq 1, \\
 & & & \forall 1 \leq k < n, (i_k \leq i_{k+1} \Rightarrow (s_k \in \mathbf{W} \text{ or } (s_k = \uparrow^t \text{ and } s_{k+1} \in \mathbf{W}))), \\
 & & & n \geq 1 \Rightarrow s_1 \in \mathbf{W} \\
c_2 & ::= & X[s_1]_{i_1} \dots [s_n]_{i_n} & \text{where } n \geq 0, \forall 1 \leq k < n, i_k \geq 1, \\
 & & & \forall 1 \leq k < n, (i_k \leq i_{k+1} \Rightarrow (s_k \in \mathbf{W} \text{ or } (s_k = \uparrow^t \text{ and } s_{k+1} \in \mathbf{W}))) \\
s & ::= & M/ \mid x \mid \uparrow^k & \text{where } k \geq 0
\end{array}
$$

Definition 6.8.8 *We call $\lambda\omega_e$-syntactic normal forms the terms $NS_{\lambda\omega_e}$ generated by the following syntax with start symbol N:*

$$
\begin{array}{lcll}
N & ::= & cN_1 \dots N_n \mid \lambda N & \text{where } n \geq 0 \\
c & ::= & c_1 \mid c_2 & \\
c_1 & ::= & m[s_1]_{i_1} \dots [s_n]_{i_n} & \text{where } m \geq 1, n \geq 0, \forall 1 \leq k \leq n, i_k \geq 1, \\
 & & & \forall 1 \leq k < n, (i_k \leq i_{k+1} \Rightarrow (s_k \in \mathbf{W} \text{ or } (s_k = \uparrow^t \text{ and } s_{k+1} \in \mathbf{W}))), \\
 & & & n \geq 1 \Rightarrow s_1 \in \mathbf{W} \\
c_2 & ::= & X[s_1]_{i_1} \dots [s_n]_{i_n} & \text{where } n \geq 0, \forall 1 \leq k < n, i_k \geq 1, \\
 & & & \forall 1 \leq k < n, (i_k \leq i_{k+1} \Rightarrow (s_k \in \mathbf{W} \text{ or } (s_k = \uparrow^t \text{ and } s_{k+1} \in \mathbf{W}))) \\
s & ::= & N/ \mid x \mid \uparrow^k & \text{where } k \geq 0
\end{array}
$$

Lemma 6.8.9 *The ω_e-syntactic normal forms are exactly the ω_e-normal forms.*

PROOF: We prove $NS_{\omega_e} \subseteq NF_{\omega_e}$ by checking that in each clause no rhs term contains any ω_e-redex.

Now we prove $NF_{\omega_e} \subseteq NS_{\omega_e}$. Let $t \in NF_{\omega_e}$. We prove that $t \in NS_{\omega_e}$ by induction on t. If $t = n$, it is clear. The same if $t = X$, $t = \lambda b$ or $t = t_1 t_2$. If t is a closure, then let $t = u[s_1]_{\mathfrak{B}_1} \dots [s_n]_{\mathfrak{B}_n}$, where u is not a closure. Then u cannot be λv, nor $t_1 t_2$, otherwise t would not be an ω_e-nf. It can only be a de Bruijn index or a meta-variable. In either case, t is generated by the c_1 or c_2 clause respectively, and in each case the conditions should hold or else t would not be an ω_e-nf. □

Lemma 6.8.10 *The $\lambda\omega_e$-syntactic normal forms are exactly the $\lambda\omega_e$-normal forms.*

PROOF: The proofs of both inclusions are analogous to the ones given in the previous Lemma. □

We also give grammars for the set of $\lambda\omega'_e$ open terms in ω'_e-normal form and for the set of $\lambda\omega'_e$ open terms in $\lambda\omega'_e$-normal form, specifying conditions in some of their rules.

Definition 6.8.11 *We call ω'_e-syntactic normal forms the terms $NS_{\omega'_e}$ generated by the following syntax with start symbol M:*

$$
\begin{array}{lcll}
M & ::= & M_1 \dots M_n \mid c \mid \lambda M & \text{where } n \geq 1 \\
c & ::= & c_1 \mid c_2 & \\
c_1 & ::= & 1[s_1]_{i_1} \dots [s_n]_{i_n} & \text{where } n \geq 0, \forall 1 \leq k \leq n, i_k \geq 1, \\
 & & & \forall 1 \leq k < n, (i_k \leq i_{k+1} \Rightarrow (s_k \in \mathbf{W} \text{ or } (s_k = \uparrow^t \text{ and } s_{k+1} \in \mathbf{W}))), \\
 & & & n \geq 1 \Rightarrow (i_1 = 1 \text{ and } (s_1 \in \mathbf{W} \text{ or } s_1 = \uparrow^t,\ t > 0)) \\
c_2 & ::= & X[s_1]_{i_1} \dots [s_n]_{i_n} & \text{where } n \geq 0, \forall 1 \leq k \leq n, i_k \geq 1, \\
 & & & \forall 1 \leq k < n, (i_k \leq i_{k+1} \Rightarrow (s_k \in \mathbf{W} \text{ or } (s_k = \uparrow^t \text{ and } s_{k+1} \in \mathbf{W}))) \\
s & ::= & M/ \mid x \mid \uparrow^k & \text{where } k \geq 0
\end{array}
$$

Definition 6.8.12 *We call $\lambda\omega'_e$-syntactic normal forms the terms $NS_{\lambda\omega'_e}$ generated by the following syntax with start symbol N:*

$$
\begin{array}{lcll}
N & ::= & cN_1 \dots N_n \mid \lambda N & \text{where } n \geq 0 \\
c & ::= & c_1 \mid c_2 & \\
c_1 & ::= & 1[s_1]_{i_1} \dots [s_n]_{i_n} & \text{where } n \geq 0, \forall 1 \leq k \leq n, i_k \geq 1, \\
 & & & \forall 1 \leq k < n, (i_k \leq i_{k+1} \Rightarrow (s_k \in \mathbf{W} \text{ or } (s_k = \uparrow^t \text{ and } s_{k+1} \in \mathbf{W}))), \\
 & & & n \geq 1 \Rightarrow (i_1 = 1 \text{ and } (s_1 \in \mathbf{W} \text{ or } s_1 = \uparrow^t,\ t > 0)) \\
c_2 & ::= & X[s_1]_{i_1} \dots [s_n]_{i_n} & \text{where } n \geq 0, \forall 1 \leq k \leq n, i_k \geq 1, \\
 & & & \forall 1 \leq k < n, (i_k \leq i_{k+1} \Rightarrow (s_k \in \mathbf{W} \text{ or } (s_k = \uparrow^t \text{ and } s_{k+1} \in \mathbf{W}))) \\
s & ::= & N/ \mid x \mid \uparrow^k & \text{where } k \geq 0
\end{array}
$$

Lemma 6.8.13 *The ω'_e-syntactic normal forms are exactly the ω'_e-normal forms.*

PROOF: We prove $NS_{\omega'_e} \subseteq NF_{\omega'_e}$ by checking that in each clause no rhs term contains any ω'_e-redex.

Now we prove $NF_{\omega'_e} \subseteq NS_{\omega'_e}$. Let $t \in NF_{\omega'_e}$. We prove that $t \in NS_{\omega'_e}$ by induction on t. If $t = 1$, it is clear. The same if $t = X$, $t = \lambda b$ or $t = t_1 t_2$. If t is a closure, then let $t = u[s_1]_{ß_1} \dots [s_n]_{ß_n}$, where u is not a closure. Then u cannot be λv, nor $t_1 t_2$, otherwise t would not be an ω_e-nf. It can only be 1 or a meta-variable. In either case, t is generated by the c_1 or c_2 clause respectively, and in each case the conditions should hold or else t would not be an ω'_e-nf. □

Lemma 6.8.14 *The $\lambda\omega'_e$-syntactic normal forms are exactly the $\lambda\omega'_e$-normal forms.*

PROOF: The proofs of both inclusions are analogous to the ones given in the previous Lemma. □

Remark that all these grammars generate all the respective normal forms including untypable normal forms (eg. such as 11).

Lemma 6.8.15 *If $a \in NF_{\omega_e}$ then $|a| \in NF_{\omega'_e}$.*

PROOF: We use induction on a. In virtue of Lemma 6.8.9, we have the following cases:

1. if $a = M_1 \dots M_n$ or $a = \lambda M$, i.e. the M clause was used, it is straightforward, since $|a| = |M_1| \dots |M_n|$ or $|a| = \lambda|M|$, so in both cases $|a|$ has no internal ω_e-redexes by the induction hypothesis.
2. analogous for the $c2$ clause.
3. analogous for the s clause.
4. for the $c1$ clause, a will have the form $m[s_1]_{i_1} \dots [s_n]_{i_n}$ where the mentioned conditions hold. Then:
   - If $n = 0$, then
     (a) either $m = 1$, then $|a| = 1$ which is an ω'_e-normal form
     (b) or $m \geq 2$, then $|a| = 1[\uparrow^{m-1}]_1$ which is also an ω'_e-normal form.
   - Else $n \geq 1$, then $s_1 \in \mathbf{W}$, and we have two cases:
     (a) if $m = 1$, $|a| = 1[|s_1|]_{i_1} \dots [|s_n|]_{i_n}$, and since $|s_1| = s_1 \in \mathbf{W}$, there are no ω'_e-redexes by the induction hypothesis, thus $|a| \in NF_{\omega'_e}$
     (b) if $m \geq 2$, $|a| = 1[\uparrow^{m-1}]_1[|s_1|]_{i_1} \dots [|s_n|]_{i_n}$, and since $|s_1| = s_1 \in \mathbf{W}$, there are no ω'_e-redexes by the induction hypothesis, thus $|a| \in NF_{\omega'_e}$.

□

Lemma 6.8.16 *If $a \in NF_{\lambda\omega_e}$ then $|a| \in NF_{\lambda\omega'_e}$.*

PROOF: We use induction on a. In virtue of Lemma 6.8.10, we have the following cases:

1. if $a = cN_1 \dots N_n$ or $a = \lambda N$, i.e. the N clause was used, it is straightforward, since $|a| = |c||N_1| \dots |N_n|$ or $|a| = \lambda|N|$, so in both cases $|a|$ has no internal $\lambda\omega_e$-redexes by the induction hypothesis.

Cases 2., 3. and 4. are analogous to items 2., 3. and 4. of the previous lemma. □

Corollary 6.8.17 (Weak normalization of typed ω'_e) *Typed ω'_e is weakly normalizing for semi-open terms.*

PROOF: Let $a \in \Lambda\omega'_{sop}$ be a typed semi-open term. By Theorem 6.5.3, a has an ω_e-normal form $\vec{\omega_e}(a)$. By Simulation and Remark 6.8.4, $a = |a| \twoheadrightarrow_{\omega'_e} |\vec{\omega_e}(a)|$. Last, $|\vec{\omega_e}(a)|$ is an ω'_e-normal form by Lemma 6.8.15. □

We will state a necessary result about li-strategies:

Lemma 6.8.18 (leftmost-innermost character preservation) *Via simulation, every li-strategy applied to a term $a \in \Lambda\omega_{sop}$ projects into a li-strategy applied to the term $|a| \in \Lambda\omega'_{sop}$.*

PROOF: We can prove that if $a \to_{\omega_e} b$ is a li-step, then $|a| \twoheadrightarrow_{\omega'_e} |b|$ is a sequence of li-steps, by induction on the position where the reduction takes place. As an illustration, we analyze the case of the σ-/-*des* rule for the case $n = j$:
$n[a/]_n \to_{\omega_e} a[\uparrow^{n-1}]_1$, so we suppose $a \in NF_{\omega_e}$ since this is a li-step, then:
$|n[a/]_n| = 1[\uparrow^{n-1}]_1[|a|/]_n$
$\to_{\omega'_e} 1[|a|/]_1[\uparrow^{n-1}]_1$
$\to_{\omega'_e} |a|[\uparrow^0]_1[\uparrow^{n-1}]_1$
$\to_{\omega'_e} |a|[\uparrow^{n-1}]_1 = |a[\uparrow^{n-1}]_1|$.
Note that all $\to_{\omega'_e}$ steps in this sequence are li, in particular the second and third steps are li because $|a| \in NF_{\omega'_e}$ by Lemma 6.8.15 thus it does not contain ω'_e-redexes.

The rest of the cases require similar or less considerations. □

Combining the previous lemmas and Theorem 6.7.10 we get

Theorem 6.8.19 (Weak normalization of typed $\lambda\omega'_e$) *Every canonical strategy for $\lambda\omega'_e$ with li ω'_e-steps is strongly normalizing and therefore the simply typed $\lambda\omega'_e$-calculus is WN for semi-open terms.*

PROOF: Let $a \in \Lambda\omega'_{sop}$. Then, since $\Lambda\omega'_{sop} \subset \Lambda\omega_{sop}$, by Theorem 6.7.10 there exists $b \in NF_{\lambda\omega_e}$ such that $a \twoheadrightarrow_{\lambda\omega_e} b$ and this derivation is a canonical strategy. Then by Remark 6.8.4 and Lemma 6.8.18, we have that $a = |a| \twoheadrightarrow_{\lambda\omega'_e} |b|$ and this derivation is a canonical strategy, and by Lemma 6.8.16 $|b| \in NF_{\lambda\omega'_e}$, thus $a \in WN_{\lambda\omega'_e}$. □

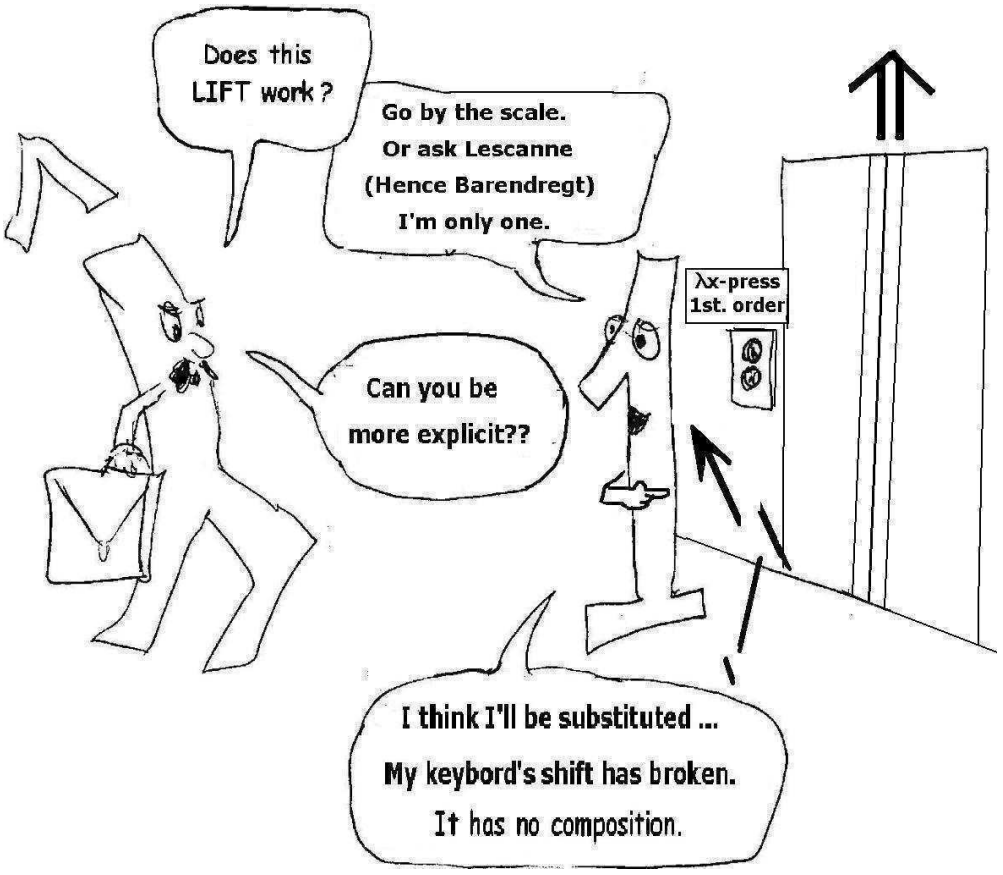
6.9 Conclusion

The main purpose of this chapter was to present a proof of weak normalization for simply typed λs_e, inspired by the technique of (37) for proving weak normalization of simply typed $\lambda\sigma$. We proved not only that typed terms are WN but we also gave a strategy for reaching the normal forms.

A main feature to emphasize is that the behavior of $\lambda\omega_e$ differs when analyzing weak normalization of open and semi-open terms. The same applies to $\lambda\omega'_e$. It is important to notice that this question for $\lambda\omega_e$ on open terms emerged when analyzing λs open terms, a calculus in which there is no distinction between semi-open and open terms since it is one-sorted.

We introduced a new calculus, $\lambda\omega'_e$, to which we transferred the same result. This calculus is closer to $\lambda\sigma$ than the calculus $\lambda\omega_e$ (which is isomorphic to $\lambda\vec{s}_e$ yet written in the $\lambda\sigma$ style), in the sense that the only de Bruijn index it uses is 1. It is a good example which shows that a calculus may not need more than a single index, if it has adequate composition rules. Thus such a new calculus has a smaller set of terms when compared to its parent. We showed that $\lambda\omega'_e$ enjoys the same good properties as $\lambda\omega_e$, by relating their respective sets of normal forms. For that purpose we provided conditional context-free grammars to describe the normal forms, this being a useful tool for performing those comparisons.

Future work includes a possible characterization of the properties that make it possible to carry over this result to other calculi. Also, it will be interesting to analyze weak normalization (possibly in a different line from (37)) for typed $\lambda\omega_e$ and $\lambda\omega'_e$ on full open terms.


Does this LIFT work?
Go by the scale.
Or ask Lescanne
(Hence Barendregt)
I'm only one.
λx-press
1st. order
Can you be more explicit??
I think I'll be substituted ...
My keybord's shift has broken.
It has no composition.

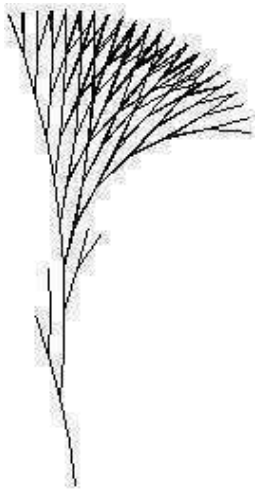


Figure 6.8: $LL(L(LX)(LL))$ after 12 left-most steps

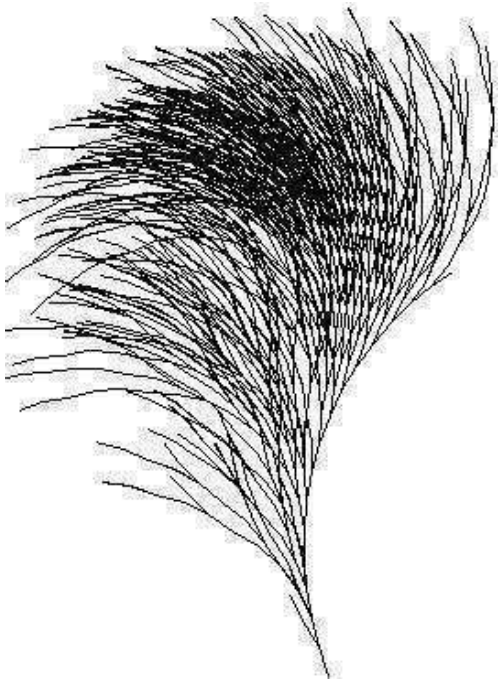


Figure 6.9: $SS(SSS)(S((S(SS))S))S(SSS)$ after 40 left-most steps

Chapter 7

A λ-calculus with constructors

In sum, in what matter soever there is place for Addition and Substraction, there also is place for Reason; and where these have no place, there Reason has nothing at all to do [...] For Reason, in this sense, is nothing but Reckoning (that is, Adding and Subtracting) of the Consequences of general names agreed upon, for the Marking and Signifying of our thoughts; I say Marking them, when we reckon by our selves; and Signifying, when we demonstrate, or approve our reckonings to other men. – T. Hobbes, *Leviathan*

Proof is the idol before whom the pure mathematician tortures himself. – A. Eddington

Roma no se construyó en un día – Anónimo

ABSTRACT We introduce a λ-calculus with constructors as an extension of classical λ-calculus with extensionality. We study all its subsystems when taking different subsets of rules. We adopt a novel approach to proving confluence, where some basic commutation lemmas are proved for some key pairs of subsystems, establishing in this way a database of commutation results, and with a computer program the commutations between new pairs of systems are inferred combinatorially until all pairs are identified as commutative or non-commutative. For this technique we formulate a set of binary closure conditions which allow to easily identify which pairs of systems commute weakly. We prove that these pairs are the same pairs that enjoy commutation. Among all 262144 pairs we prove that 26544 pairs commute (in particular 248 subsystems are confluent, and the same subsystems which are weakly confluent are confluent). We also introduce and prove a separation theorem based on syntactical disagreement.

7.1 Introduction

Lambda-calculus has been introduced by Church in the 30's (19) as a universal language to express computations of functions. Despite its remarkable simplicity, λ-calculus is rich enough to express all recursive functions. Since the rise of computers, λ-calculus has been used fruitfully as the basis of all functional programming languages, from LISP to the languages of the ML family (40; 68; 70). From the theoretical point of view, untyped λ-calculus enjoys many good properties (11), such as Church and Rosser's property expressing determinism of computations. In Logic, λ-calculus is also a fundamental tool to describe the computational contents of proofs via the Curry-Howard correspondence.

Although arbitrarily complex data structures can be encoded in the pure λ-calculus, modern functional programming languages provide primitive constructs for most data structures, for which a purely functional encoding would be inefficient. One of the most popular extensions of λ-calculus is pattern-matching on constructed values (a.k.a. variants), a problem that has been widely investigated in functional programming (40; 68; 70) and in rewriting (18; 21; 41; 42; 80).

However, introducing objects of different kinds—functions and constructed values—in the same formalism addresses the problem of their interaction. What does it mean to apply a constructed value $cM_1 \cdots M_n$ to an argument? Should the constructed value accumulate the extra argument? Or should it produce an error? Similarly, what does it mean to perform case analysis on a function?

Unfortunately, these problems are usually not addressed in the literature because they are irrelevant in a typed setting—applications go with functions, case analysis with variants. However, one should not forget that one of the reasons of the success of the λ-calculus in computer science and in logic lies in its excellent operational semantics in the untyped case. The best example is given by Böhm's separation theorem (16), that expresses that two observationally equivalent $\beta\eta$-normal λ-terms are intentionally equal. In the pure λ-calculus, $\beta\eta$-normal terms are not canonical forms because they cannot be further reduced; they are canonical forms because the computational behavior of a $\beta\eta$-normal term cannot be expressed by another $\beta\eta$-normal term.

The situation is far from being as clear when we add pattern-matching to the untyped λ-calculus. As far as we know, there is no generalization of Böhm's theorem for this kind of extension. One reason for that is that the notion of normal form is not as clear as in the pure λ-calculus, precisely because the traditional operational semantics says nothing about the computational behavior of ill-typed constructions, such as a case analysis over an abstraction.

An extended operational semantics of case analysis In this chapter, we propose an extension of the untyped λ-calculus with constructors and case analysis that fills the holes of

the traditional operational semantics. Technically, the main novelty is that we let application and case analysis (written $\{\!|\theta|\!\}.\, M$) commute via the (ill-typed[1]) reduction rule

$$(\textsc{CaseApp}) \qquad \{\!|\theta|\!\}.\, (MN) \quad \to \quad (\{\!|\theta|\!\}.\, M)N\,.$$

(Here, θ denotes a *case binding*, that is a finite map from constructors to terms.) Symmetrically, we introduce a reduction rule

$$(\textsc{CaseLam}) \qquad \{\!|\theta|\!\}.\, (\lambda x\,.\, N) \quad \to \quad \lambda x\,.\, (\{\!|\theta|\!\}.\, M) \qquad (x \notin FV(\theta))$$

to let case analysis go through abstractions. In this way, case analysis can be understood as a form of head linear explicit substitution... of constructors.

Surprisingly, the system we obtain is not only computationally sound—we will show that it is confluent (sections 7.4, 7.5 and 7.6) and conservative over the untyped $\lambda\eta$-calculus, i.e. the latter is a sub-calculus—but it also permits to decompose ML-style pattern matching (with patterns of any arity) from the construction $\{\!|\theta|\!\}.\, M$ that only performs case analysis on constant constructors.

Finally, we will show (section 7.7) a theorem of weak separation for the whole calculus, using a separation technique inspired by Böhm's (11; 16). For this reason, the formalism provides a special constant written ✠ and called the *daimon* (following Girard's terminology and notation (34)) that requests the termination of the program –something like am `exit` system call– and which will be used as the main technical device to observe normal forms and to separate them.

This chapter is based on the joint work in (8).

7.2 Syntax and reduction rules

In this section we provide syntax and rules for our calculus.

7.2.1 Syntax

The λ-calculus with constructors distinguishes two kinds of names: *variables* (written x, y, z, etc.) and *constructors* (written c, c', etc.) The set of variables and the set of constructors are written $\mathcal{V}$ and $\mathcal{C}$, respectively. In what follows, we assume that both sets $\mathcal{V}$ and $\mathcal{C}$ are denumerable and disjoint.

[1] Observe that M is treated as a function in the l.h.s. of the rule whereas it is treated as a constructed value in the r.h.s. This rule should not be confused with the rule of *commutative conversion* $(\{\!|\theta|\!\}.\, M)N = \{\!|\theta N|\!\}.\, M$ that comes from logic, a rule which is well-typed... but incompatible with the reduction rules of our calculus!

The terms (written M, N, etc.) and the case bindings (written θ, ϕ, etc.) of the λ-calculus with constructors are inductively defined as follows:

Terms	M, N	$::=$	x	(Variable)
		$\mid$	c	(Constructor)
		$\mid$	$\maltese$	(Daimon)
		$\mid$	MN	(Application)
		$\mid$	$\lambda x\,.\,M$	(Abstraction)
		$\mid$	$\{\!\{\theta\}\!\}\,.\,M$	(Case construct)
Case bindings	θ, ϕ	$::=$	$c_1 \mapsto M_1; \ldots; c_n \mapsto M_n$	$(c_i \neq c_j \text{ for } i \neq j)$

We denote the set of terms with $\Lambda_{\mathcal{C}}$, the set of case bindings with $\mathcal{B}$, and the disjoint union of $\Lambda_{\mathcal{C}}$ and $\mathcal{B}$ with $\Lambda_{\mathcal{C}}+\mathcal{B}$.

7.2.1.1 Constructor binding

Each case binding θ is formed as an finite unordered list of constructor bindings of the form $(c \mapsto M)$ whose l.h.s. are pairwise distinct. We say that a constructor c is *bound* to a term M in a case binding θ if the binding $(c \mapsto M)$ belongs to the list θ. From the definition of case bindings, it is clear that a constructor c is bound to at most one term in a given case binding θ. When there is no such binding, we say that the constructor c is *unbound* in θ.

For $\theta = \{c_i \mapsto M_i\}_{i=1,\ldots,n}$, we write $|\theta| = n$.

We also introduce an (external) operation of *composition* between two case bindings θ and ϕ, which is written $\theta \circ \phi$ and defined by:

$$\theta \circ (c_1 \mapsto M_1; \ldots; c_n \mapsto M_n) \quad \equiv \quad c_1 \mapsto \{\!\{\theta\}\!\}\,.\,M_1; \ldots; c_n \mapsto \{\!\{\theta\}\!\}\,.\,M_n$$

(where $\phi \equiv (c_1 \mapsto M_1; \ldots; c_n \mapsto M_n)$). Notice that this operation is not syntactically associative, since:

$$(\theta \circ \phi) \circ (c_1 \mapsto M_1; \ldots; c_n \mapsto M_n) \quad \equiv \quad c_1 \mapsto \{\!\{\theta \circ \phi\}\!\}\,.\,M_1; \ldots; c_n \mapsto \{\!\{\theta \circ \phi\}\!\}\,.\,M_n$$

whereas

$$\theta \circ (\phi \circ (c_1 \mapsto M_1; \ldots; c_n \mapsto M_n)) \quad \equiv \quad c_1 \mapsto \{\!\{\theta\}\!\}\,.\,\{\!\{\phi\}\!\}\,.\,M_1; \ldots; c_n \mapsto \{\!\{\theta\}\!\}\,.\,\{\!\{\phi\}\!\}\,.\,M_n\,.$$

However, composition of case bindings only makes sense in the presence of the case conversion reduction rule $\{\!\{\theta\}\!\}\,.\,\{\!\{\phi\}\!\}\,.\,M \to \{\!\{\theta \circ \phi\}\!\}\,.\,M$ (see 7.2.2), for which both right hand sides above are convertible.

7.2.1.2 Free variables and meta-substitution

The notions of bound and free occurrences of a variable are defined as expected. The set of free variables of a term M (resp. a case binding θ) is written $FV(M)$ (resp. $FV(\theta)$), where for $\theta = \{c_i \mapsto M_i\}_{i=1,\ldots,n}$, we define $FV(\theta) = FV(M_1) \cup \ldots FV(M_n)$ and $FV(\{\!\{\theta\}\!\}\,.\,M) = FV(\theta) \cup$

$FV(M)$. As in the (ordinary) λ-calculus, terms are considered up to α-conversion (i.e. up to a renaming of bound variables). Notice that the renaming policy of the λ-calculus with constructors is strictly the same as in the λ-calculus: it only affects (bound) *variable names*, but leaves *constructor names* unchanged.

The external substitution operation of the λ-calculus, written $M\{x := N\}$, is extended to the λ-calculus with constructors as expected. The same operation is also defined for case bindings (notation: $\theta\{x := N\}$) in the obvious way: for $\theta = \{c_i \mapsto M_i\}_{i=1,\ldots,n}$, we define $\theta\{\mathtt{x} := N\} = \{c_i \mapsto M_i\{\mathtt{x} := N\}\}_{i=1,\ldots,n}$, and $(\{\!|\theta|\!\}.\, M)\{\mathtt{x} := N\} = \{\!|\theta\{\mathtt{x} := N\}|\!\}.\, M\{\mathtt{x} := N\}$.

7.2.2 Reduction rules

The λ-calculus with constructors has 9 different reduction rules that are divided into four reduction groups, that are given in Fig. 7.1.

Beta-reduction				
AppLam (AL)	$(\lambda x\,.\,M)N$	$\to$	$M\{x := N\}$	
AppDai (AD)	$\maltese\, N$	$\to$	$\maltese$	
Eta-reduction				
LamApp (LA)	$\lambda x\,.\,Mx$	$\to$	M	$(x \notin FV(M))$
LamDai (LD)	$\lambda x\,.\,\maltese$	$\to$	$\maltese$	
Case propagation				
CaseCons (CO)	$\{\!|\theta|\!\}.\, c$	$\to$	M	$((c \mapsto M) \in \theta)$
CaseDai (CD)	$\{\!|\theta|\!\}.\, \maltese$	$\to$	$\maltese$	
CaseApp (CA)	$\{\!|\theta|\!\}.\, (MN)$	$\to$	$(\{\!|\theta|\!\}.\, M)N$	
CaseLam (CL)	$\{\!|\theta|\!\}.\, \lambda x\,.\,M$	$\to$	$\lambda x\,.\,\{\!|\theta|\!\}.\, M$	$(x \notin FV(\theta))$
Case conversion				
CaseCase (CC)	$\{\!|\theta|\!\}.\, \{\!|\phi|\!\}.\, M$	$\to$	$\{\!|\theta \circ \phi|\!\}.\, M$	

Figure 7.1: Reduction rules of the λ-calculus with constructors

In what follows, our interest will focus not only on the reduction induced by the 9 reduction rules (taken together), but also on all the subsystems formed by any combination of these 9 rules. We call $\lambda\mathcal{B}_{\mathcal{C}}$-calculus the calculus generated by the 9 rules given in Figure 7.1, and we call $\mathcal{B}_{\mathcal{C}}$-calculus the calculus generated by all the rules except AppLam.

Let R be a binary relation over terms. Taking the contextual closure of R naturally defines two binary relations: one over terms, and another one over case bindings, both being written $\rightarrow_R$ and called the *one step R-reduction* (over terms, over case bindings). The reflexive and transitive closure of $\rightarrow_R$ (over terms, over case bindings) is written $\xrightarrow{*}_R$ as well as $\twoheadrightarrow_R$ and called the *R-reduction relation* (over terms, over case bindings). We will also use the reflexive closure of $\rightarrow_R$ (over terms, over case bindings) which will be written $\rightarrow_R^=$. Finally, the reflexive, symmetric and transitive closure of $\rightarrow_R$ (over terms, over case bindings) is written $\simeq_R$ and called the *R-equality relation* (over terms, over case bindings).

As we will see in section 7.6, the reduction relation defined by these 9 rules (taken together) enjoys the Church-Rosser property, i.e. the $\lambda\mathcal{B}_{\mathcal{C}}$-calculus is confluent.

We conclude this section with some lemmas giving satisfactory conditions like in classical λ-calculus, which will be useful later on.

Lemma 7.2.1 *Let $P, Q \in \Lambda_{\mathcal{C}} + \mathcal{B}$. If $P \rightarrow_{\lambda\mathcal{B}_{\mathcal{C}}} Q$, then $FV(Q) \subseteq FV(P)$.*

Proof: By induction on P analyzing each one of the rules. Note that for the CaseCase-rule, if $\phi = \{c_i \mapsto M_i\}_{i=1,\ldots,n}$, then
$FV(\theta \circ \phi) = FV(\{c_i \mapsto \{\!|\theta|\!\}.\, M_i\}_{i=1,\ldots,n}) = FV(\theta) \cup FV(M_1) \cup \cdots \cup FV(M_n)$. □

Lemma 7.2.2 *For all terms and case bindings M, for every term P and variable x, if $x \notin FV(M)$ then $M\{\mathtt{x} := P\} = M$.*

Proof: By induction on M. □

Lemma 7.2.3 *Let $P, Q \in \Lambda_{\mathcal{C}} + \mathcal{B}$ and y a variable. Then $FV(P\{\mathtt{y} := Q\}) \subseteq FV(P) - \{y\} \cup FV(Q)$.*

Proof: By induction on P much in the same way as for classical λ-calculus. For case binder $\theta = \{c_i \mapsto M_i\}_{i=1,\ldots,n}$, we have
$FV(\theta) = \cup_{i=1,\ldots,n} FV(M_i) =_{IH} \cup_{i=1,\ldots,n}(FV(M_i) - \{y\} \cup FV(Q))$
$= (\cup_{i=1,\ldots,n} FV(M_i)) - \{y\} \cup FV(Q) = FV(\theta) - \{y\} \cup FV(Q)$ □

Now we can state the Meta-substitution Lemma for $\lambda\mathcal{B}_{\mathcal{C}}$ extending the result of classical λ-calculus.

Lemma 7.2.4 (Meta-substitution Lemma) *For all terms and case bindings M, for all terms P, Q and variables x, y, if $y \neq x \notin FV(Q)$ then we have that $M\{\mathtt{x} := P\}\{\mathtt{y} := Q\} = M\{\mathtt{y} := Q\}\{\mathtt{x} := P\{\mathtt{y} := Q\}\}$.*

Proof: By induction on M.

- If $M = x$ we have by Lemma 7.2.2 that $P\{\mathtt{y} := Q\} = P\{\mathtt{y} := Q\}$.

- If $M = y$, we have that $Q = Q$ since by hypothesis $x \notin FV(Q)$.
- If $M = z(\neq x, y)$, we have that $z = z$.
- If $M = c$ a constructor, we have that $c = c$.
- If $M = \Lightning$, we have that $\Lightning = \Lightning$.
- If $M = M_1 M_2$, $M\{\mathtt{x} := P\}\{\mathtt{y} := Q\} =$
  $M_1\{\mathtt{x} := P\}\{\mathtt{y} := Q\}M_2\{\mathtt{x} := P\}\{\mathtt{y} := Q\} =_{IH}$
  $M_1\{\mathtt{y} := Q\}\{\mathtt{x} := P\{\mathtt{y} := Q\}\}M_2\{\mathtt{y} := Q\}\{\mathtt{x} := P\{\mathtt{y} := Q\}\} =$
  $M\{\mathtt{y} := Q\}\{\mathtt{x} := P\{\mathtt{y} := Q\}\}$.
- If $M = \lambda z.M_1$, $M\{\mathtt{x} := P\}\{\mathtt{y} := Q\} =$
  $\lambda z.M_1\{\mathtt{x} := P\}\{\mathtt{y} := Q\} =_{IH}$
  $\lambda z.M_1\{\mathtt{y} := Q\}\{\mathtt{x} := P\{\mathtt{y} := Q\}\} =$
  $M\{\mathtt{y} := Q\}\{\mathtt{x} := P\{\mathtt{y} := Q\}\}$.
- If $M = \theta = \{c_i \mapsto M_i\}_{i=1,\ldots,n}$, $\theta\{\mathtt{x} := P\}\{\mathtt{y} := Q\} =$
  $\{c_i \mapsto M_i\{\mathtt{x} := P\}\{\mathtt{y} := Q\}\}_{i=1,\ldots,n} =_{IH}$
  $\{c_i \mapsto M_i\{\mathtt{y} := Q\}\{\mathtt{x} := P\{\mathtt{y} := Q\}\}\}_{i=1,\ldots,n} =$
  $\theta\{\mathtt{y} := Q\}\{\mathtt{x} := P\{\mathtt{y} := Q\}\}$.
- If $M = \{\!|\theta|\!\}.\, M_1$, $M\{\mathtt{x} := P\}\{\mathtt{y} := Q\} =$
  $\{\!|\theta\{\mathtt{x} := P\}\{\mathtt{y} := Q\}|\!\}.\, M_1\{\mathtt{x} := P\}\{\mathtt{y} := Q\} =_{IH}$
  $\{\!|\theta\{\mathtt{y} := Q\}\{\mathtt{x} := P\{\mathtt{y} := Q\}\}|\!\}.\, M_1\{\mathtt{y} := Q\}\{\mathtt{x} := P\{\mathtt{y} := Q\}\} =$
  $M\{\mathtt{y} := Q\}\{\mathtt{x} := P\{\mathtt{y} := Q\}\}$.

□

7.2.3 The need of ✠

As a side note, the ✠ functionality cannot be mimicked by a regular term. Even restricting to classical λ-calculus (which is a subsystem of $\lambda\mathcal{B}_{\mathcal{C}}$), there is no term $D \in \Lambda$ such that $\lambda x.D \xrightarrow{*}_\beta D$, simply because $\lambda x.D \to_\beta D$ implies $D = \lambda x.D'$ with $D \to_\beta D'$ and then one obtains by induction on the derivation that $\lambda x.D = D$ which is an absurd equality. The other property required for D, which is $DM \xrightarrow{*}_\beta D$ for every M, is indeed possible in λ-calculus taking $D = YK$ where Y is the Turing fixpoint combinator and $K = \lambda xy.x$. But then D would not have β-normal form, and this is not adequate for the idea of ✠ which is to stop immediately (very far from not having a normal form). This plainly motivates the addition of ✠ to the language. Such a special term will be exploited in the Separation Theorem near the end of the chapter. We also noticed that in principle ✠ can be added to calculi of explicit substitution preserving its main idea.

7.3 Strong Normalization of the $\mathcal{B}_{\mathcal{C}}$-calculus

In this section we prove that the substitution calculus $\mathcal{B}_{\mathcal{C}}$ enjoys strong normalization (SN). This is a key result which will be useful later on for the proofs of confluence.

Proposition 7.3.1 (SN of $\mathcal{B}_{\mathcal{C}}$-calculus) *The $\mathcal{B}_{\mathcal{C}}$-calculus is SN, i.e. there are no infinite derivations of the form $M_1 \to_{\mathcal{B}_{\mathcal{C}}} M_2 \to_{\mathcal{B}_{\mathcal{C}}} \ldots$.*

Proof: Let $h : \Lambda_{\mathcal{C}} + \mathcal{B} \to \mathbb{N}$ be defined by mutual induction for terms by

$$\begin{array}{lcl} h(x) = h(c) = h(\maltese) & = & 1 \\ h(MN) & = & h(M) + h(N) \\ h(\lambda x.M) & = & h(M) + 1 \\ h(\{\!|\theta|\!\}.\, M) & = & h(\theta) + (|\theta| + 2)h(M) \end{array}$$

and for case bindings by

$$h(\{c_i \mapsto M_i\}_{i=1,\ldots,n}) \;=\; \textstyle\sum_{i=1}^{n} h(M_i)$$

It is routine to check that for all $P, Q \in \Lambda_{\mathcal{C}} + \mathcal{B}$, if $P \to_{\mathcal{B}_{\mathcal{C}}} Q$ then $h(P) > h(Q)$. □

Corollary 7.3.2 (SN of $\lambda\mathcal{B}_{\mathcal{C}}$-subsystems) *Let s be a subsystem of $\lambda\mathcal{B}_{\mathcal{C}}$-calculus, given by a subset of its rules. Then s is SN iff* AppLam $\notin s$.

This important result will be used extensively within the proof of confluence in section 7.6.

7.4 Preliminary definitions and commutation results

We give some general commutation results, which we will use extensively hereinafter.

Lemma 7.4.1 *Let A, B, C be ARSs.*

1. *If $A//B$ and $A//C$ then $A//B + C$.*
2. *If $A//_w B$ and $A//_w C$ then $A//_w B + C$.*

Proof:

1. Given a divergence from an element x for relation A versus $B + C$, the diagram can be closed by tiling with the corresponding commutation diagrams for $A//B$ and $A//C$.
2. Immediate from the definition since a single-step divergence for relations A versus $B + C$ may be of case A versus B or case A versus C.

□

We generalize Newman's Lemma (see chapter 1) for the case of two reduction relations, as follows.

Lemma 7.4.2 *Let A, B be ARSs such that $A//_w B$ and $A + B$ is SN. Then, $A//B$.*

PROOF: Similar to Newman's Lemma, by well-founded induction. □

7.5 General closure conditions

We are interested in proving the CR property for the λ-calculus with constructors, as well as several of its subsystems.

In order to prepare for the confluence proof, we show the critical pairs of λ-calculus with constructors in Figures 7.2 and 7.3. In Figure 7.2 we show the critical pairs which require other rules to close, while in Figure 7.3 we show the critical pairs that close without using other rules. Another distinction between both sets of critical pairs is that the second set involves the CASECASE-rule.

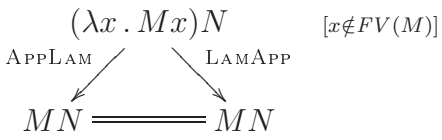
We will introduce the *binary closure conditions* (BCC), which apply to pairs of systems, as well as the *closure conditions* (CC) which apply to systems. Two systems will weakly commute iff they satisfy the BCC. The form of these conditions is the following: $r_1 \in s_1$ and $r_2 \in s_2$ then $r_3 \in s_1$, for r_i rules and s_i systems. Therefore it will be quite simple to check any BCC on any pair of systems.

Definition 7.5.1 (Closure conditions) — *We say that a subsystem s formed by a subset of the nine rules given in Figure 7.1 enjoys the* closure conditions *(CC) if it is closed under the following 6 conditions:*

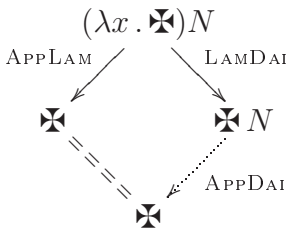
(CC1)	APPLAM $\in s$,	LAMDAI $\in s$	$\vdash$	APPDAI $\in s$
(CC2)	LAMAPP $\in s$,	APPDAI $\in s$	$\vdash$	LAMDAI $\in s$
(CC3)	CASEAPP $\in s$,	APPLAM $\in s$	$\vdash$	CASELAM $\in s$
(CC4)	CASEAPP $\in s$,	APPDAI $\in s$	$\vdash$	CASEDAI $\in s$
(CC5)	CASELAM $\in s$,	LAMAPP $\in s$	$\vdash$	CASEAPP $\in s$
(CC6)	CASELAM $\in s$,	LAMDAI $\in s$	$\vdash$	CASEDAI $\in s$

Definition 7.5.2 (Binary closure conditions) — *We say that a pair of subsystems (s_1, s_2)*

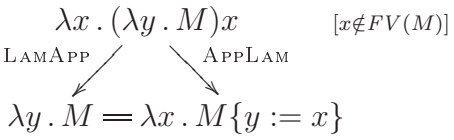
(1) AppLam/LamApp



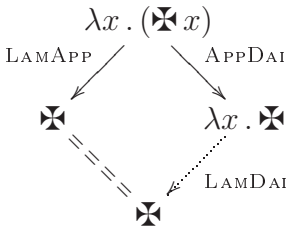
(2) AppLam/LamDai



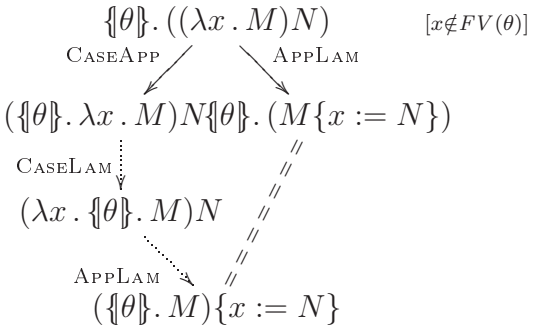
(3) LamApp/AppLam



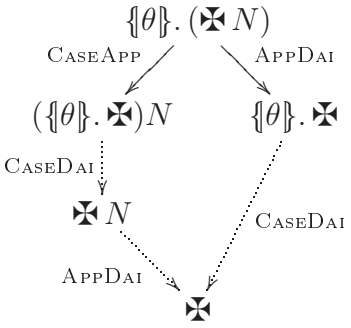
(4) LamApp/AppDai



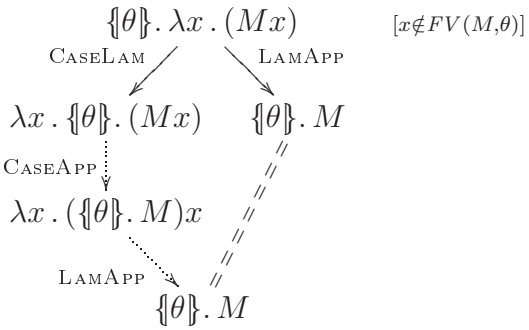
(5) CaseApp/AppLam



(6) CaseApp/AppDai



(7) CaseLam/LamApp



(8) CaseLam/LamDai

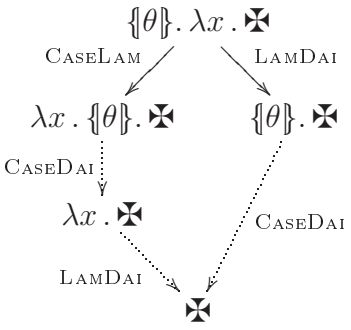


Figure 7.2: Critical pairs 1–8 (/13)

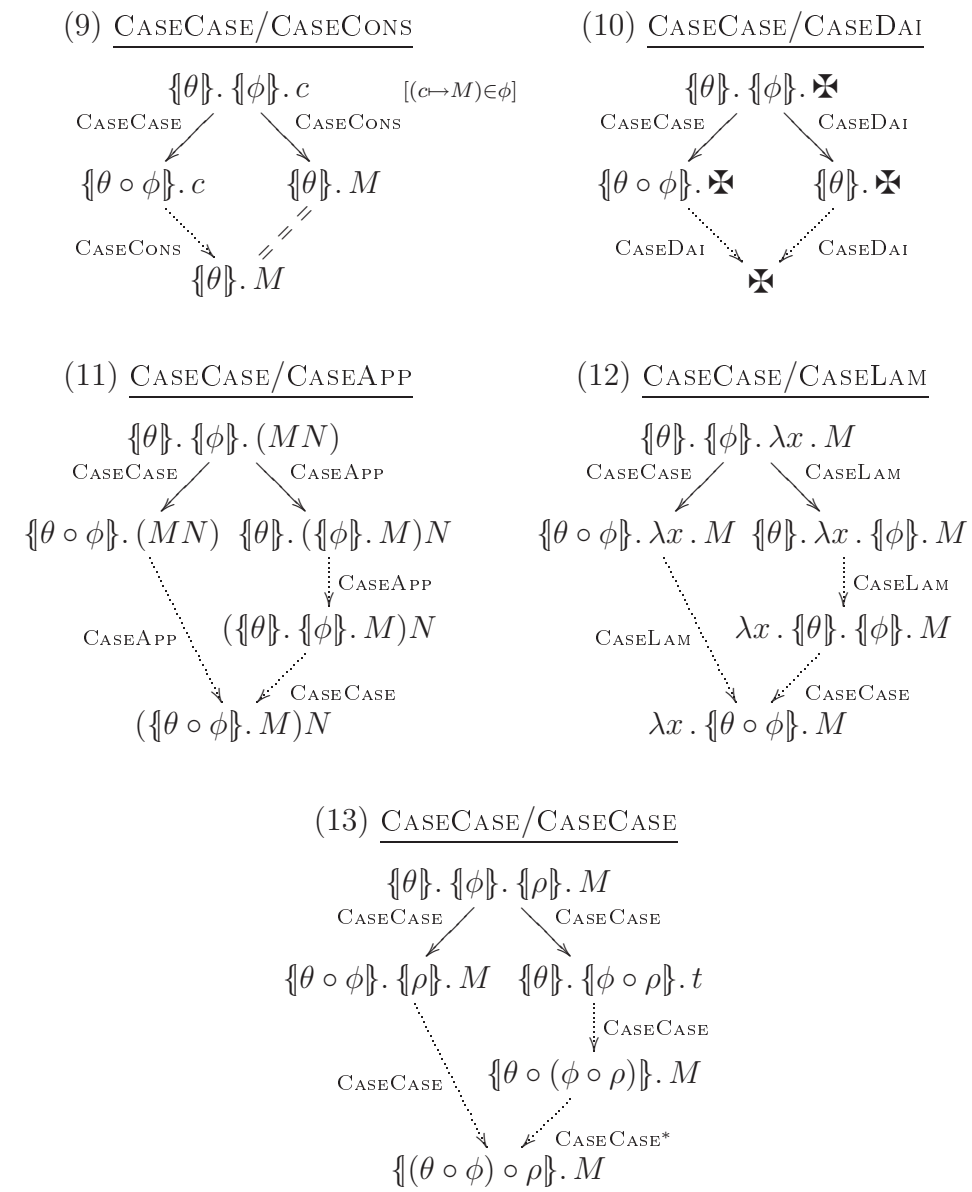


Figure 7.3: Critical pairs 9–13 (/13)

enjoys the binary closure conditions *(BCC) if it is closed under the following 9 conditions:*

(BCC1)	$\textsc{AppLam} \in s_1,$	$\textsc{LamDai} \in s_2$	$\vdash$	$\textsc{AppDai} \in s_1$
(BCC2)	$\textsc{LamApp} \in s_1,$	$\textsc{AppDai} \in s_2$	$\vdash$	$\textsc{LamDai} \in s_1$
(BCC3)	$\textsc{CaseApp} \in s_1,$	$\textsc{AppLam} \in s_2$	$\vdash$	$\textsc{CaseLam} \in s_2$
(BCC4)	$\textsc{CaseApp} \in s_1,$	$\textsc{AppDai} \in s_2$	$\vdash$	$\textsc{CaseDai} \in (s_1 \cap s_2)$
(BCC5)	$\textsc{CaseLam} \in s_1,$	$\textsc{LamApp} \in s_2$	$\vdash$	$\textsc{CaseApp} \in s_2$
(BCC6)	$\textsc{CaseLam} \in s_1,$	$\textsc{LamDai} \in s_2$	$\vdash$	$\textsc{CaseDai} \in (s_1 \cap s_2)$
(BCC7)	$\textsc{CaseCase} \in s_1,$	$\textsc{CaseDai} \in s_2$	$\vdash$	$\textsc{CaseDai} \in s_1$
(BCC8)	$\textsc{CaseCase} \in s_1,$	$\textsc{CaseApp} \in s_2$	$\vdash$	$\textsc{CaseApp} \in s_1$
(BCC9)	$\textsc{CaseCase} \in s_1,$	$\textsc{CaseLam} \in s_2$	$\vdash$	$\textsc{CaseLam} \in s_1$

as well as under the 9 symmetric conditions obtained by interchanging s_1 with s_2.

Remark 7.5.3 *A subsystem s fulfills the closure conditions iff the pair (s, s) fulfills the binary closure conditions.*

Proof: Obvious. Take $s_1 = s_2 = s$ in Definition 7.5.2. □

Theorem 7.5.4 (General weak commutation) — *Given two subsystems s_1 and s_2, the following propositions are equivalent:*

1. *The pair (s_1, s_2) fulfills the binary closure conditions;*
2. *The reduction relations s_1 and s_2 weakly commute;*

Proof: (2)⇒(1) holds because, when considering arbitrary terms, the critical pairs can only be closed as shown in Figures 7.2 and 7.3, therefore the BCC must hold[1]. (1)⇒(2) it is easy to verify that the binary closure conditions allow to close the critical pairs (see Figures 7.2 and 7.3). □

Corollary 7.5.5 (General weak confluence) — *Given a subsystem s, the following propositions are equivalent:*

1. *The subsystem s fulfills the closure conditions;*
2. *The s-reduction is locally confluent;*

Proof: Immediate using Theorem 7.5.4 and Remark 7.5.3. □

[1] The best way of checking this is to think that in those figures the letters M and N may be replaced by fresh free variables, thus the only way to close a critical pair is by using the indicated rules in each case

7.6 The Church-Rosser property

This section is devoted to the proof of the commutation of several pairs of subsystems. Actually, to prove that $s_1 //_w s_2$ iff $s_1 // s_2$ for all pairs of subsystems s_1, s_2. As a corollary, the confluence of the whole system will be obtained.

Unless otherwise specified, all commutation and weak commutation results will be stated not only for terms but also for case bindings. In particular, we are interested in the CR property for $\lambda\mathcal{B}_{\mathcal{C}}$, i.e. the entire set of rules.

In studying the confluence of our system and subsystems, we adopt a novel approach given by the following method of "computer assisted proof":

We develop a computer program which

1. given an initial table of commutation and weak commutation results (i.e. two sets of pairs of subsystems of the $\lambda\mathcal{B}_{\mathcal{C}}$-calculus)

2. given the information about subsets of rules which are SN (for us, all subsystems which exclude the AppLam-rule), and

3. given the binary closure conditions (in Section 7.5)

infers all additional commutation results combinatorially using Lemmas 7.4.1 and 7.4.2.

The program implements the algorithm in Figure 7.4.

The main idea of this algorithm is the following:

1. it starts with an initial database of results, to be checked "by hand" (i.e. they have to be proved)

2. the main routine infers additional commutation results using lemmas 7.4.1 and 7.4.2

3. it adds the inferred lemmas to the database, and cycle until no new commutation result can be added

So we end up with a complete table of commutation (and therefore confluence) results among (all) subsystems of the $\lambda\mathcal{B}_{\mathcal{C}}$-calculus.

What is also interesting is that in this way, during the cycle which adds commutation entries to the table, the program can output not only the inferred commutation results but also the new lemmas which will be required to be proved by hand (and to be added to the database afterwards) in order to continue with the iterations. Even one could also add a *what-if* result temporarily, and find out which other lemmas should be proved after that supposition.

The natural advantage of this approach is that one can prove just the (minimal set of) required lemmas. This technique for proving confluence gives an alternative proof to the Interpretation

– main algorithm

Algorithm commutingPairs;
uses the arrays WC and C indexed by pairs of subsystems

begin

initialize $C(\bullet, \bullet)$ with
true where stated by *database lemmas*, and
false elsewhere;

copy $C(\bullet, \bullet)$ into $WC(\bullet, \bullet)$;

for each pair of subsystems (s_1, s_2)
if $(s_1, s_2) \vdash BCC$ **then**
$WC(s_1, s_2) :=$ **true**; – Theorem 7.5.4

for each pair of subsystems (s_1, s_2)
if $WC(s_1, s_2)$ and $isSN(s_1 \cup s_2)$ **then**
$C(s_1, s_2) :=$ **true**; – Lemma 7.4.2

repeat
for each 3-uple of subsystems (s_1, s_2, s_3)
if $C(s_1, s_2)$ and $C(s_1, s_3)$ **then**
if $C(s_1, s_2 \cup s_3) =$ **false then** *change it to* **true**;
– Lemma 7.4.1
until no more changes are performed;

end;

– to determine if a $\lambda\mathcal{B}_{\mathcal{C}}$-subsystem is SN

$isSN$ (s a subsystem);
begin;
if AppLam $\in s$ **then return false;**
else return true;
– Corollary 7.3.2
end;

Figure 7.4: The commutation inference algorithm

Method (75) with the benefit that commutation and confluence results are obtained for the subsystems as well.

Thus the following subsections of this section are devoted to prove "by hand" the initial database commutation results, which will be given by commutation diagrams.

We will prove several lemmas by induction on the structure of a term. In each of these proofs, if there is no overlap the corresponding diagram can be easily closed, otherwise the critical pairs presented in Figures 7.2 and 7.3 will guide the proof.

The following subsections thus provide:

1. Preservation by meta-substitution
2. Commutation of AppLam with AppDai, CaseCons, CaseDai and CaseLam
3. Commutation of AppLam+CaseLam with CaseApp
4. Commutation of CaseCase with AppLam
5. Commutation of AppLam with AppLam
6. Commutation of AppLam+AppDai with LamDai
7. Commutation of AppLam with LamApp
8. Commutation of AppLam+AppDai with LamApp+LamDai
9. Commutation of AppLam+CaseLam with LamApp+CaseApp
10. Commutation of AppLam+AppDai+CaseLam+CaseDai with LamApp+LamDai+ CaseApp+CaseDai, and
11. General Commutation and Confluence

7.6.1 Preservation by meta-substitution

We now prove important basic lemmas which relate meta-substitution with the $\mathcal{B}_{\mathcal{C}}$-rules, to be used in the commutation diagrams of the subsequent lemmas. All these lemmas state that meta-substitution is preserved under the $\lambda\mathcal{B}_{\mathcal{C}}$-rules.

We formulate these lemmas in a generic way, i.e. treating every calculus rule generically, in order to have a single statement for each one of them.

We will first need the distribution of meta-substitution over composition.

Remark 7.6.1 *For all case bindings θ, ϕ, for every term P and variable y,*
$(\theta \circ \phi)\{\mathtt{y} := P\} = \theta\{\mathtt{y} := P\} \circ \phi\{\mathtt{y} := P\}$

PROOF: Let $\phi = \{c_i \mapsto M_i\}_{i=1,\ldots,n}$.
Then $(\theta \circ \phi)\{\mathtt{y} := P\} = \{c_i \mapsto \{\!|\theta|\!\}.\, M_i\}_{i=1,\ldots,n}\{\mathtt{y} := P\}$
$= \{c_i \mapsto \{\!|\theta\{\mathtt{y} := P\}|\!\}.\, M_i\{\mathtt{y} := P\}_{i=1,\ldots,n}\}$
$= \theta\{\mathtt{y} := P\} \circ \{c_i \mapsto M_i\{\mathtt{y} := P\}\}_{i=1,\ldots,n}$
$= \theta\{\mathtt{y} := P\} \circ \phi\{\mathtt{y} := P\}$ □

Now we can give the generic

Lemma 7.6.2 *Let R be any rule in the set* {AppLam, AppDai, LamApp, LamDai, CaseLam, CaseApp, CaseDai, CaseCons, CaseCase}.

1. *For all terms and case bindings M, N, for every term P and variable y, if $M \to_R N$ then $M\{\mathtt{y} := P\} \to_R N\{\mathtt{y} := P\}$.*
2. *For all terms and case bindings M, for all terms P, Q and variable y, if $P \to_R Q$, then $M\{\mathtt{y} := P\} \stackrel{*}{\to}_R M\{\mathtt{y} := Q\}$*

PROOF:

1. By a straightforward induction on M.
   - If $M = x, y, \maltese, c$, the result holds vacuously.
   - If the reduction is at the root, we have the following possibilities:
     - $R =$ AppLam, then $M = (\lambda x.Q)R \to_{\text{AppLam}} Q\{\mathtt{x} := R\} = N$,
       then $M\{\mathtt{y} := P\} = (\lambda x.Q\{\mathtt{y} := P\})R\{\mathtt{y} := P\} \to_{\text{AppLam}}$
       $Q\{\mathtt{y} := P\}\{\mathtt{x} := R\{\mathtt{y} := P\}\} = Q\{\mathtt{x} := R\}\{\mathtt{y} := P\}$ (by Lemma 7.2.4)
       $= N\{\mathtt{y} := P\}$.
     - $R =$ AppDai, then $M = \maltese M_1 \to_{\text{AppDai}} \maltese = N$,
       then $M\{\mathtt{y} := P\} = \maltese M_1\{\mathtt{y} := P\} \to_{\text{AppDai}} \maltese = N\{\mathtt{y} := P\}$.
     - $R =$ LamApp, then $M = (\lambda x.Qx) \to_{\text{LamApp}} Q$ where $x \notin FV(Q)$ and $x \neq y$,
       then $M\{\mathtt{y} := P\} = \lambda x.Q\{\mathtt{y} := P\}x$
       $\to_{\text{LamApp}} Q\{\mathtt{y} := P\}$ using Lemma 7.2.3, since by the free variable convention
       $x \notin FV(P)$ and $x \notin FV(Q)$ thus
       $x \notin FV(Q) - \{y\} \cup FV(P)$.
     - $R =$ LamDai, then $M = \lambda x.\maltese \to_{\text{LamDai}} \maltese = N$,
       then $M\{\mathtt{y} := P\} = \lambda x.\maltese\{\mathtt{y} := P\} = \lambda x.\maltese \to_{\text{LamDai}} \maltese$
       $= N\{\mathtt{y} := P\}$.
     - $R =$ CaseLam, then $M = \{\!|\theta|\!\}.\, \lambda x.M_1 \to_{\text{CaseLam}} \lambda x.\{\!|\theta|\!\}.\, M_1$
       $= N$, then $M\{\mathtt{y} := P\} = \{\!|\theta\{\mathtt{y} := P\}|\!\}.\, (\lambda x.M_1)\{\mathtt{y} := P\}$
       $= \{\!|\theta\{\mathtt{y} := P\}|\!\}.\, \lambda x.M_1\{\mathtt{y} := P\}$
       $\to_{\text{CaseLam}} \lambda x.\{\!|\theta\{\mathtt{y} := p\}|\!\}.\, M_1\{\mathtt{y} := P\} = N\{\mathtt{y} := P\}$.

- $R = \textsc{CaseApp}$, then $M = \{\!|\theta|\!\}.\,(M_1 M_2) \to_{\textsc{CaseApp}} \{\!|\theta|\!\}.\,M_1 M_2 = N$,
  then $M\{\mathtt{y} := P\} = \{\!|\theta\{\mathtt{y} := P\}|\!\}.\,(M_1\{\mathtt{y} := P\} M_2\{\mathtt{y} := P\})$
  $\to_{\textsc{CaseApp}} \{\!|\theta\{\mathtt{y} := P\}|\!\}.\,M_1\{\mathtt{y} := P\} M_2\{\mathtt{y} := P\} = N\{\mathtt{y} := P\}$.
- $R = \textsc{CaseDai}$, then $M = \{\!|\theta|\!\}.\,\maltese \to_{\textsc{CaseDai}} \maltese = N$,
  then $M\{\mathtt{y} := P\} = \{\!|\theta\{\mathtt{y} := P\}|\!\}.\,\maltese\{\mathtt{y} := P\}$
  $= \{\!|\theta\{\mathtt{y} := P\}|\!\}.\,\maltese \to_{\textsc{CaseDai}} \maltese = N\{\mathtt{y} := P\}$.
- $R = \textsc{CaseCons}$, then $M = \{\!|\{c_i \mapsto M_i\}_{i=1,\ldots,n}|\!\}.\,c_j$
  $\to_{\textsc{CaseCons}} M_j = N$, then $M\{\mathtt{y} := P\}$
  $= \{\!|\{c_i \mapsto M_i\}_{i=1,\ldots,n}\{\mathtt{y} := P\}|\!\}.\,c_j\{\mathtt{y} := P\}$
  $= \{\!|\{c_i \mapsto M_i\{\mathtt{y} := P\}\}_{i=1,\ldots,n}|\!\}.\,c_j$
  $\to_{\textsc{CaseCons}} M_j\{\mathtt{y} := P\} = N\{\mathtt{y} := P\}$.
- $R = \textsc{CaseCase}$, then $M = \{\!|\theta|\!\}.\,\{\!|\phi|\!\}.\,Q \to_{\textsc{CaseCase}} \{\!|\theta \circ \phi|\!\}.\,Q = N$,
  then $M\{\mathtt{y} := P\} = \{\!|\theta\{\mathtt{y} := P\}|\!\}.\,\{\!|\phi\{\mathtt{y} := P\}|\!\}.\,Q\{\mathtt{y} := P\}$
  $\to_{\textsc{CaseCase}} \{\!|\theta\{\mathtt{y} := P\} \circ \phi\{\mathtt{y} := P\}|\!\}.\,Q\{\mathtt{y} := P\}$
  $= \{\!|(\theta \circ \phi)\{\mathtt{y} := P\}|\!\}.\,Q\{\mathtt{y} := P\}$ (by Remark 7.6.1)
  $= (\{\!|\theta \circ \phi|\!\}.\,Q)\{\mathtt{y} := P\} = N\{\mathtt{y} := P\}$.

- If $M = M_1 M_2 \to_R N_1 M_2$ with $M_1 \to_R N_1$, then
  $M\{\mathtt{y} := P\} = M_1\{\mathtt{y} := P\} M_2\{\mathtt{y} := P\}$
  $\to_R^{IH} N_1\{\mathtt{y} := P\} M_2\{\mathtt{y} := P\} = N\{\mathtt{y} := P\}$.
- If $M = M_1 M_2 \to_R M_1 N_2$ with $M_2 \to_R N_2$, analogous to the previous case.
- If $M = \lambda x.M_1 \to_R \lambda x.N_1$ with $M_1 \to_R N_1$, then
  $M\{\mathtt{y} := P\} = \lambda x.M_1\{\mathtt{y} := P\} \to_R^{IH} \lambda x.N_1\{\mathtt{y} := P\} = N\{\mathtt{y} := P\}$ using the free variable convention.
- If $M = \theta = \{c_i \mapsto M_i\}_{i=1,\ldots,n} \to_R \{c_i \mapsto N_i\}_{i=1,\ldots,n} = \phi$ where $M_i = N_i$ for $i \neq j$ for some $1 \leq j \leq n$ and $M_j \to_R N_j$, then
  $M\{\mathtt{y} := P\} = \{c_i \mapsto M_i\{\mathtt{y} := P\}\}_{i=1,\ldots,n}$
  $\to_R^{IH} \{c_i \mapsto N_i\{\mathtt{y} := P\}\}_{i=1,\ldots,n} = \theta\{\mathtt{y} := P\}$.
- If $M = \{\!|\theta|\!\}.\,M_1 \to_R \{\!|\phi|\!\}.\,M_1 = N$ and $\theta \to_R \phi$, then
  $M\{\mathtt{y} := P\} = \{\!|\theta\{\mathtt{y} := P\}|\!\}.\,M_1\{\mathtt{y} := P\}$
  $\to_R^{IH} \{\!|\phi\{\mathtt{y} := P\}|\!\}.\,M_1\{\mathtt{y} := P\}$
  $= N\{\mathtt{y} := P\}$.
- If $M = \{\!|\theta|\!\}.\,M_1 \to_R \{\!|\theta|\!\}.\,N_1 = N$ with $M_1 \to_R N_1$, then
  $M\{\mathtt{y} := P\} = \{\!|\theta\{\mathtt{y} := P\}|\!\}.\,M_1\{\mathtt{y} := P\}$
  $\to_R^{IH} \{\!|\theta\{\mathtt{y} := P\}|\!\}.\,N_1\{\mathtt{y} := P\} = N\{\mathtt{y} := P\}$.

2. By induction on M.

- If $M = y$ we have $P \to_R Q$ (1 step).

- If $M = x \neq y$ we have $x \xrightarrow{*}_R x$ (0 steps).
- If $M = \maltese$ we have $\maltese \xrightarrow{*}_R \maltese$ (0 steps).
- If $M = c$ we have $c \xrightarrow{*}_R c$ (0 steps).
- If $M = M_1 M_2$, $M\{\mathtt{y} := P\} = M_1\{\mathtt{y} := P\}M_2\{\mathtt{y} := P\}$ $\xrightarrow{*}_R^{IH} M_1\{\mathtt{y} := Q\}M_2\{\mathtt{y} := Q\} = M\{\mathtt{y} := Q\}$.
- If $M = \lambda x.M_1$, $M\{\mathtt{y} := P\} = \lambda x.M_1\{\mathtt{y} := P\}$ $\xrightarrow{*}_R^{IH} \lambda x.M_1\{\mathtt{y} := Q\} = M\{\mathtt{y} := Q\}$.
- If $M = \theta = \{c_i \mapsto M_i\}_{i=1,\ldots,n}$, $M\{\mathtt{y} := P\} = \{c_i \mapsto M_i\{\mathtt{y} := P\}\}_{i=1,\ldots,n}$ $\xrightarrow{*}_R^{IH} \{c_i \mapsto M_i\{\mathtt{y} := Q\}\}_{i=1,\ldots,n} = \theta\{\mathtt{y} := Q\}$.
- If $M = \{\!|\theta|\!\}.\, M_1$, $M\{\mathtt{y} := P\} = \{\!|\theta\{\mathtt{y} := P\}|\!\}.\, M_1\{\mathtt{y} := P\}$ $\xrightarrow{*}_R^{IH} \{\!|\theta\{\mathtt{y} := Q\}|\!\}.\, M_1\{\mathtt{y} := Q\} = M\{\mathtt{y} := Q\}$.

□

Actually we need the extension to many-step reductions.

Corollary 7.6.3 *Let R be any $\lambda\mathcal{B}_\mathcal{C}$-rule.*

1. *For all terms and case bindings M, N, for every term P and variable y, if $M \xrightarrow{*}_R N$ then $M\{\mathtt{y} := P\} \xrightarrow{*}_R N\{\mathtt{y} := P\}$.*
2. *For all terms and case bindings M, for all terms P, Q and variable y, if $P \xrightarrow{*}_R Q$, then $M\{\mathtt{y} := P\} \xrightarrow{*}_R M\{\mathtt{y} := Q\}$*

Proof:

1. By Lemma 7.6.2(1) and induction on the length of the derivation $M \xrightarrow{*}_R N$.
2. By Lemma 7.6.2(2) and induction on the length of the derivation $P \xrightarrow{*}_R Q$.

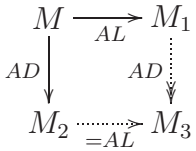
□

7.6.2 Commutation of AL with AD, CO, CD and CL

We prove the necessary commutation lemmas for these one-rule systems.

Lemma 7.6.4 AppLam//AppDai

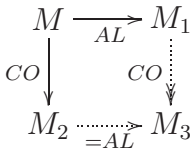
PROOF: We prove that the following diagram holds for terms and case bindings (where we denote both sorts by using the letter M):



It is done by induction on M. There is no critical pair. It uses Corollary 7.6.3 (1) and (2). Note that the $=$ subscript appears at the AppLam-reduction since AppDai may erase redexes, and a many-step AppDai-derivation appears because AppLam may erase or duplicate redexes. Since they strongly commute, by the Commutation Lemma (Lemma 1.3.7) they commute. □

Lemma 7.6.5 AppLam//CaseCons

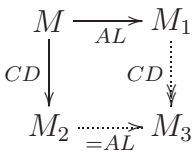
PROOF: We prove that the following diagram holds for terms and case bindings:



It is done by induction on M. There is no critical pair. It uses Corollary 7.6.3 (1) and (2). Note that the $=$ subscript appears at the AppLam-reduction since CaseCons may erase redexes, and a many-step CaseCons-derivation appears because AppLam may erase or duplicate redexes. Since they strongly commute, by the Commutation Lemma they commute. □

Lemma 7.6.6 AppLam//CaseDai

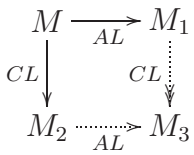
PROOF: We prove that the following diagram holds for terms and case bindings:



It is done by induction on M. There is no critical pair. It uses Corollary 7.6.3 (1) and (2). Note that the $=$ subscript appears at the AppLam-reduction since CaseDai may erase redexes, and a many-step CaseDai-derivation appears because AppLam may erase or duplicate redexes. Since they strongly commute, by the Commutation Lemma they commute. □

Lemma 7.6.7 AppLam//CaseLam

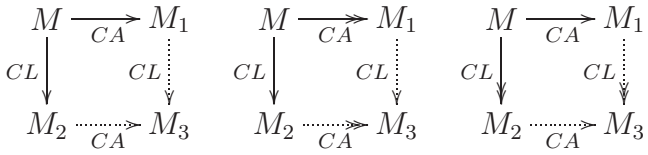
PROOF: We prove that the following diagram holds for terms and case bindings:



It is done by induction on M. There is no critical pair. It uses Corollary 7.6.3 (1) and (2). Note that the many-step CASELAM-derivation appears because APPLAM may erase or duplicate redexes. Since they strongly commute, by the Commutation Lemma they commute. □

7.6.3 Commutation of $AL + CL$ with CA

Lemma 7.6.8 *The following diagrams hold for terms and case bindings:*

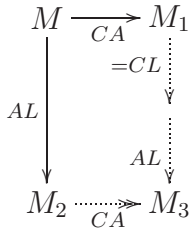


PROOF:

1. By induction on M. There is no critical pair.
2. By induction on the length of the CA-derivation using item (1).
3. By induction on the length of the CL-derivation using item (1).

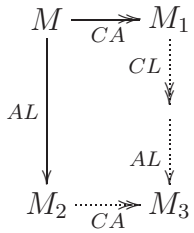
□

Lemma 7.6.9 *The following diagram holds for terms and case bindings:*

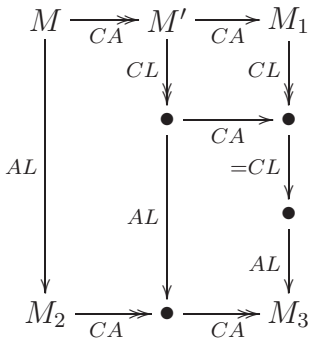


PROOF: By induction on M. There is one critical pair, which closes according to the diagram. It uses Corollary 7.6.3 (1) and (2). Note that APPLAM may erase or duplicate a redex thus the many-step CASEAPP-derivation at the bottom. □

Lemma 7.6.10 *The following diagram holds for terms and case bindings:*



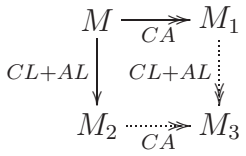
PROOF: By induction on the length of the CASEAPP-derivation, using Lemma 7.6.9 and Lemma 7.6.8(2). The picture is:



where the left rectangle can be closed by IH. □

Lemma 7.6.11 APPLAM + CASELAM//CASEAPP

PROOF: We show first that the following diagram holds:



using Lemma 7.6.8(3) and Lemma 7.6.10. Then we conclude by induction on the derivation $M \xrightarrow{*}_{CL+AL} M_2$. □

7.6.4 Commutation of CC with AL

To show commutation of CASECASE with APPLAM we will use the parallel reduction technique. We define for this a parallel version of APPLAM, which will also help in proving confluence of the latter.

Definition 7.6.12 *We define the parallel* APPLAM *reduction as follows:*

$$\frac{M \Rightarrow M' \quad N \Rightarrow N'}{(\lambda x.M)N \Rightarrow M'\{\mathtt{x} := N'\}}\,(pAppLam) \qquad \frac{}{M \Rightarrow M}\,(pRef)$$

$$\frac{M \Rightarrow M'}{\lambda x.M \Rightarrow \lambda x.M'}\,(pLam) \qquad \frac{M_i \Rightarrow M_i', i = 1,\ldots,n}{\{c_i \mapsto M_i\}_{i=1,\ldots,n} \Rightarrow \{c_i \mapsto M_i'\}_{i=1,\ldots,n}}\,(pCB)$$

$$\frac{M \Rightarrow M' \quad N \Rightarrow N'}{MN \Rightarrow M'N'}\,(pApp) \qquad \frac{M \Rightarrow M' \quad \theta \Rightarrow \theta'}{\{\!|\theta|\!\}.\,M \Rightarrow \{\!|\theta'|\!\}.\,M'}\,(pCase)$$

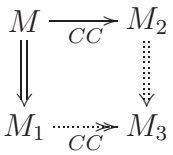
Now we state Proposition 7.6.13 which provides a sequence of easy but useful structural properties to be used afterwards:

Proposition 7.6.13 (Structure preservation by CaseCase and $\Rightarrow$) *For all terms M, M_1, M_2, N, case bindings θ, ϕ and variable x*

1. *if $\lambda x.M \to_{\text{CaseCase}} N$ then there exists M' such that $N = \lambda x.M'$ with $M \to_{\text{CaseCase}} M'$*
2. *if $M_1 M_2 \to_{\text{CaseCase}} N$ then*
   - *either there exists M_1' such that $N = M_1' M_2$ with $M_1 \to_{\text{CaseCase}} M_1'$*
   - *or there exists M_2' such that $N = M_1 M_2'$ with $M_2 \to_{\text{CaseCase}} M_2'$*
3. *if $\theta \to_{\text{CaseCase}} \phi$ then there exist $M_1, \ldots, M_n, N$, and constructors $c_1, \ldots, c_n$, with $n \geq 1$ such that $\theta = \{c_i \mapsto M_i\}_{i=1,\ldots,n}$ and there exists $1 \leq j \leq n$ such that $\phi = \{c_i \mapsto N_i\}_{i=1,\ldots,n}$ with $N_i = M_i$ for $i \neq j$ and $M_j \to_{\text{CaseCase}} N_j$*
4. *if $\{\!|\theta|\!\}.\, M \to_{\text{CaseCase}} N$ then*
   - *either there exists θ' such that $N = \{\!|\theta'|\!\}.\, M$ with $\theta \to_{\text{CaseCase}} \theta'$*
   - *or there exists M' such that $N = \{\!|\theta|\!\}.\, M'$ with $M \to_{\text{CaseCase}} M'$*
   - *or there exists ϕ, M' such that $M = \{\!|\phi|\!\}.\, M'$ and $N = \{\!|\theta \circ \phi|\!\}.\, M'$*
5. *if $\lambda x.M \Rightarrow N$ then there exists M' such that $N = \lambda x.M'$ with $M \Rightarrow M'$*
6. *if $M_1 M_2 \Rightarrow N$ then*
   - *either there exist M_1', M_2' such that $N = M_1' M_2'$ with $M_1 \Rightarrow M_1'$ and $M_2 \Rightarrow M_2'$*
   - *or there exists P, P', M_2' and a variable y such that $M_1 = \lambda y.P$ and $N = P'\{\mathtt{y} := M_2'\}$ with $P \Rightarrow P'$ and $M_2 \Rightarrow M_2'$*
7. *if $\theta \Rightarrow \phi$ then there exist $M_1, \ldots, M_n, N$, and constructors $c_1, \ldots, c_n$, with $n \geq 1$ such that $\theta = \{c_i \mapsto M_i\}_{i=1,\ldots,n}$ and $\phi = \{c_i \mapsto N_i\}_{i=1,\ldots,n}$ with $M_i \Rightarrow N_i$ for $1 \leq i \leq n$*
8. *if $\{\!|\theta|\!\}.\, M \Rightarrow N$ then there exists θ', M' such that $N = \{\!|\theta'|\!\}.\, M'$ with $\theta \Rightarrow \theta'$ and $M \Rightarrow M'$*

Proof: All items are proved by induction on M or θ, the same way as in the classical λ-calculus. □

Having defined parallel AppLam reduction (Definition 7.6.12) we will prove that it commutes strongly with CaseCase (Lemma 7.6.17), which will imply that AppLam commutes with Case-Case (Corollary 7.6.20). In other words, we will prove that the following diagram holds for terms and case bindings:



We need the following technical lemmata. All these lemmas will be formulated for $\Lambda_{\mathfrak{C}}$ terms as well as for $\mathcal{B}$ case bindings. Proofs will be by simultaneous induction on terms and case bindings. In what follows, we also use Barendregt's *free variable convention* whenever necessary.

Similar to Lemma 7.6.2, meta-substitution is preserved under parallel AppLam reduction.

Lemma 7.6.14 *For every term and case binding M, for all terms P, Q and every variable y, if $P \Rightarrow Q$, then $M\{\mathtt{y} := P\} \Rightarrow M\{\mathtt{y} := Q\}$*

Proof: By induction on M.

- If $M = y$, then we have $P \Rightarrow Q$.
- If $M = x \neq y$, then by reflexivity $x \Rightarrow x$.
- If $M = \maltese$, then by reflexivity $\maltese \Rightarrow \maltese$.
- If $M = c$ a constructor, then by reflexivity $c \Rightarrow c$.
- If $M = M_1 M_2$, then by IH and pApp $M\{\mathtt{y} := P\} = M_1\{\mathtt{y} := P\}M_2\{\mathtt{y} := P\}$
  $\Rightarrow M_1\{\mathtt{y} := Q\}M_2\{\mathtt{y} := Q\} = M\{\mathtt{y} := Q\}$.
- If $M = \lambda x.M_1$, then by IH and pLam $M\{\mathtt{y} := P\} = \lambda x.M_1\{\mathtt{y} := P\}$
  $\Rightarrow \lambda x.M_1\{\mathtt{y} := Q\} = M\{\mathtt{y} := Q\}$.
- If $M = \theta = \{c_i \mapsto M_i\}_{i=1,\ldots,n}$, then by IH and (pCB)
  $M\{\mathtt{y} := P\} = \{c_i \mapsto M_i\{\mathtt{y} := P\}\}_{i=1,\ldots,n}$
  $\Rightarrow \{c_i \mapsto M_i\{\mathtt{y} := Q\}\}_{i=1,\ldots,n} = \theta\{\mathtt{y} := Q\}$.
- If $M = \{\!|\theta|\!\}.\, N$, then by IH and pCase
  $M\{\mathtt{y} := P\} = \{\!|\theta\{\mathtt{y} := P\}|\!\}.\, N\{\mathtt{y} := P\}$
  $\Rightarrow \{\!|\theta\{\mathtt{y} := Q\}|\!\}.\, N\{\mathtt{y} := Q\} = M\{\mathtt{y} := Q\}$.

□

The following is a generalization of the previous lemma:

Lemma 7.6.15 *For all terms and case bindings P, Q, for all terms R, S and every variable y, if $P \Rightarrow Q$ and $R \Rightarrow S$, then $P\{\mathtt{y} := R\} \Rightarrow Q\{\mathtt{y} := S\}$*

PROOF: By induction on the derivation of $P \Rightarrow Q$.

- if (pRef) was applied, $P = Q$, then by Lemma 7.6.14 $P\{\mathtt{y} := R\} \Rightarrow Q\{\mathtt{y} := S\}$.
- for (pAppLam), $P = (\lambda x.M)N$, $Q = M'\{\mathtt{x} := N'\}$ with $M \Rightarrow M'$ and $N \Rightarrow N'$, then $((\lambda x.M)N)\{\mathtt{y} := R\} = (\lambda x.M\{\mathtt{y} := R\})N\{\mathtt{y} := R\}$. By IH, $M\{\mathtt{y} := R\} \Rightarrow M'\{\mathtt{y} := S\}$ and $N\{\mathtt{y} := R\} \Rightarrow N'\{\mathtt{y} := S\}$, thus
  $(\lambda x.M\{\mathtt{y} := R\})N\{\mathtt{y} := R\} \Rightarrow M'\{\mathtt{y} := S\}\{\mathtt{x} := N'\{\mathtt{y} := S\}\}$
  $= M'\{\mathtt{x} := N'\}\{\mathtt{y} := S\}$ (by Lemma 7.2.4)
  $= Q\{\mathtt{y} := S\}$ since x is fresh by the free variable convention.
- for (pApp), $(MN)\{\mathtt{y} := R\} = M\{\mathtt{y} := R\}N\{\mathtt{y} := R\} \Rightarrow^{IH}$
  $M'\{\mathtt{y} := S\}N'\{\mathtt{y} := S\} = M'N'\{\mathtt{y} := S\}$
- for (pLam), $(\lambda x.M)\{\mathtt{y} := R\} = \lambda x.M\{\mathtt{y} := R\} \Rightarrow^{IH} \lambda x.M'\{\mathtt{y} := S\} = (\lambda x.M')\{\mathtt{y} := S\}$
- for (pCB), $\{c_i \mapsto M_i\}_{i=1,\ldots,n}\{\mathtt{y} := R\} = \{c_i \mapsto M_i\{\mathtt{y} := R\}\}_{i=1,\ldots,n} \Rightarrow^{IH} \{c_i \mapsto M_i'\{\mathtt{y} := S\}\}_{i=1,\ldots,n} = \{c_i \mapsto M_i'\}_{i=1,\ldots,n}\{\mathtt{y} := S\}$
- for (pCase), $\{\!|\theta|\!\}.\, N\{\mathtt{y} := R\} = \{\!|\theta\{\mathtt{y} := R\}|\!\}.\, N\{\mathtt{y} := R\} \Rightarrow^{IH}$
  $\{\!|\theta'\{\mathtt{y} := S\}|\!\}.\, N'\{\mathtt{y} := S\} = \{\!|\theta'|\!\}.\, N'\{\mathtt{y} := S\}$

□

We still need the following

Lemma 7.6.16 *For all case bindings $\theta, \theta', \phi, \phi'$, if $\theta \Rightarrow \theta'$ and $\phi \Rightarrow \phi'$ then $\theta \circ \phi \Rightarrow \theta' \circ \phi'$.*

PROOF: Let $\phi = \{d_i \mapsto N_i\}_{i=1,\ldots,n} \Rightarrow \{d_i \mapsto N_i'\}_{i=1,\ldots,n} = \phi'$ with $N_i \Rightarrow N_i'$ for all $1 \leq i \leq n$. Then $\theta \circ \phi = \{d_i \mapsto \{\!|\theta|\!\}.\, N_i\}_{i=1,\ldots,n} \Rightarrow \{d_i \mapsto \{\!|\theta'|\!\}.\, N_i'\}_{i=1,\ldots,n} = \theta' \circ \phi'$. □

Then we get

Lemma 7.6.17 (parallel AppLam and CaseCase strong commutation) *For all terms M, M_1, M_2, if $M \Rightarrow M_1$ and $M \to_{\text{CaseCase}} M_2$, then there exists M_3 such that $M_1 \overset{*}{\to}_{\text{CaseCase}} M_3$ and $M_2 \Rightarrow M_3$. And analogously for case bindings. In other words, the following diagrams hold:*

$$\begin{array}{ccc} M & \xrightarrow[CC]{} & M_2 \\ \Downarrow & & \vdots \\ M_1 & \xrightarrow[CC]{} & M_3 \end{array} \qquad \begin{array}{ccc} \theta & \xrightarrow[CC]{} & \theta_2 \\ \Downarrow & & \vdots \\ \theta_1 & \xrightarrow[CC]{} & \theta_3 \end{array}$$

PROOF: We reason by induction on the derivation $M \Rightarrow M_1$. We have the following cases:

1. (pRef) was applied, with $M = M_1$, take $M_3 = M_2$.

2. (pAppLam) was applied, with $M = (\lambda x.P)Q$, $M_1 = P'\{\mathtt{x} := Q'\}$, $P \Rightarrow P'$,$Q \Rightarrow Q'$, so by Proposition 7.6.13 (2) and (1) we have that either

   - $M_2 = (\lambda x.P'')Q$ with $P \to_{\text{CaseCase}} P''$, that is

   $$\begin{array}{ccc} (\lambda x.P)Q & \xrightarrow[CC]{} & (\lambda x.P'')Q \\ \Downarrow & & \\ P'\{\mathtt{x} := Q'\} & & \end{array}$$

   By IH the following diagram holds for some P''':

   $$\begin{array}{ccc} P & \xrightarrow[CC]{} & P'' \\ \Downarrow & & \vdots \\ P' & \xrightarrow[CC]{} & P''' \end{array}$$

   By Corollary 7.6.3, $P'\{\mathtt{x} := Q'\} \xrightarrow{*}_{\text{CaseCase}} P'''\{\mathtt{x} := Q'\}$, and since $P'' \Rightarrow P'''$ and $Q \Rightarrow Q'$, $(\lambda x.P'')Q \Rightarrow P'''\{\mathtt{x} := Q'\}$ so the diagram is closed taking $M_3 = P'''\{\mathtt{x} := Q'\}$.

   - or $M_2 = (\lambda x.P)Q''$ with $Q \to_{\text{CaseCase}} Q''$, in which case

   $$\begin{array}{ccc} (\lambda x.P)Q & \xrightarrow[CC]{} & (\lambda x.P)Q'' \\ \Downarrow & & \\ P'\{\mathtt{x} := Q'\} & & \end{array}$$

   By IH the following diagram holds for some Q''':

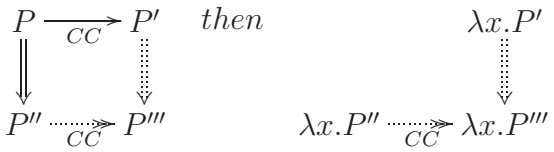
   $$\begin{array}{ccc} Q & \xrightarrow[CC]{} & Q'' \\ \Downarrow & & \vdots \\ Q' & \xrightarrow[CC]{} & Q''' \end{array}$$

   By Corollary 7.6.3, $P'\{\mathtt{x} := Q'\} \xrightarrow{*}_{\text{CaseCase}} P'\{\mathtt{x} := Q'''\}$, and since $P \Rightarrow P'$ and $Q'' \Rightarrow Q'''$, $(\lambda x.P)Q'' \Rightarrow P'\{\mathtt{x} := Q'''\}$ so the diagram is closed taking $M_3 = P'\{\mathtt{x} := Q'''\}$.

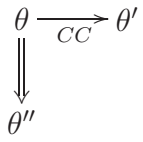
3. (pLam) was applied, with $M = \lambda x.P$, $M_1 = \lambda x.P''$ and $P \Rightarrow P''$, in which case by Proposition 7.6.13 (1) there exists P' such that

   $$\begin{array}{ccc} \lambda x.P & \xrightarrow[CC]{} & \lambda x.P' \\ \Downarrow & & \\ \lambda x.P'' & & \end{array}$$

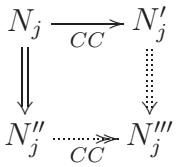
with $P \to_{\text{CaseCase}} P'$. By IH we have



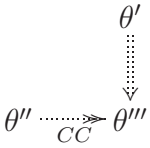
4. (pCB) was applied, with $\theta = \{c_i \mapsto N_i\}_{i=1,\ldots,n}$, $\theta'' = \{c_i \mapsto N_i''\}_{i=1,\ldots,n}$, $N_i \Rightarrow N_i''$ for $1 \leq i \leq n$, in which case by Proposition 7.6.13 (3)



so for some j we have by IH the diagram

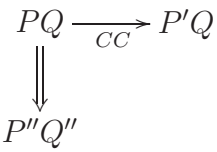


then taking $\theta''' = \{c_i \mapsto N_i'''\}_{i=1,\ldots,n}$ with $N_i''' = N_i''$ if $i \neq j$

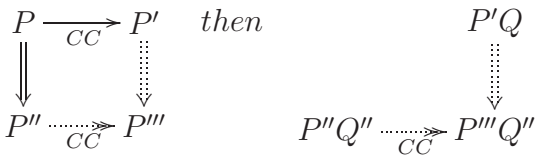


5. (pApp) was applied, with $M = PQ$, $M_1 = P''Q''$, so by Proposition 7.6.13 (2) we have that either

   - $M_2 = P'Q$ with $P \to_{\text{CaseCase}} P'$, in which case



     with $Q \Rightarrow Q''$ and $P \Rightarrow P''$, and by IH we have the diagram



   - or $M_2 = PQ'$ with $Q \to_{\text{CaseCase}} Q'$, in which case the diagram is closed analogously.

6. (pCase) was applied, with $M = \{\!|\theta|\!\}.Q$, $M_1 = \{\!|\theta'|\!\}.Q'$, $\theta \Rightarrow \theta'$ and $Q \Rightarrow Q'$, so by Proposition 7.6.13 (4) we have that either

   - CaseCase was applied at the root, i.e. $Q = \{\!|\phi|\!\}.P$, so by Proposition 7.6.13 (5) $Q' = \{\!|\phi'|\!\}.P'$ with $\phi \Rightarrow \phi'$, $P \Rightarrow P'$ and we have

$$\begin{array}{ccc} \{\!|\theta|\!\}.\{\!|\phi|\!\}.P & \xrightarrow[CC]{} & \{\!|\theta \circ \phi|\!\}.P \\ \Downarrow & & \\ \{\!|\theta'|\!\}.\{\!|\phi'|\!\}.P' & & \end{array}$$

     By Lemma 7.6.16, $\theta \circ \phi \Rightarrow \theta' \circ \phi'$, thus $\{\!|\theta \circ \phi|\!\}.P \Rightarrow \{\!|\theta' \circ \phi'|\!\}.P'$. Since $\{\!|\theta'|\!\}.\{\!|\phi'|\!\}.P' \rightarrow_{\text{CaseCase}} \{\!|\theta' \circ \phi'|\!\}.P'$, the diagram is closed.

   - or an internal CaseCase was applied, then either
     - $\theta \rightarrow_{\text{CaseCase}} \theta''$, so we have

$$\begin{array}{ccc} \{\!|\theta|\!\}.Q & \xrightarrow[CC]{} & \{\!|\theta''|\!\}.Q \\ \Downarrow & & \\ \{\!|\theta'|\!\}.Q' & & \end{array}$$

       and by IH we have

$$\begin{array}{ccc} \theta & \xrightarrow[CC]{} & \theta'' \\ \Downarrow & & \vdots \\ \theta' & \xrightarrow[CC]{} & \theta''' \end{array} \quad \textit{then} \quad \begin{array}{ccc} & & \{\!|\theta''|\!\}.Q \\ & & \vdots \\ \{\!|\theta'|\!\}.Q' & \xrightarrow[CC]{} & \{\!|\theta'''|\!\}.Q' \end{array}$$

     - or $Q \rightarrow_{\text{CaseCase}} Q''$, and the diagram is closed analogously.

□

As a generalization of the results in Corollary 7.6.3 for AppLam we have:

Corollary 7.6.18 *For all terms and case bindings M, N, for all terms P, Q and variable y, if $M \xrightarrow{*}_{\text{AppLam}} N$ and $P \xrightarrow{*}_{\text{AppLam}} Q$, then $M\{\mathtt{y} := P\} \xrightarrow{*}_{\text{AppLam}} N\{\mathtt{y} := Q\}$*

Proof: Since $M \xrightarrow{*}_{\text{AppLam}} N$, by Corollary 7.6.3 (1) we have that
$M\{\mathtt{y} := P\} \xrightarrow{*}_{\text{AppLam}} N\{\mathtt{y} := P\}$, then by Corollary 7.6.3 (2)
$M\{\mathtt{y} := P\} \xrightarrow{*}_{\text{AppLam}} N\{\mathtt{y} := Q\}$. □

Lemma 7.6.19 $\xrightarrow{*}_{\text{AppLam}} \;=\; \overset{*}{\Rightarrow}$

Proof: We first show that $\rightarrow_{\text{AppLam}} \subseteq \Rightarrow$ (it essentially uses the reflexivity rule). For this we show that $M \rightarrow_{\text{AppLam}} N$ implies $M \Rightarrow N$ by induction on M.

- If $M = x, c, \maltese$, the result holds vacuously.
- If the reduction takes place at the root, say $(\lambda x.P)Q \to_{\textsc{AppLam}} P\{\mathtt{x} := Q\}$, then $(\lambda x.P)Q \Rightarrow P\{\mathtt{x} := Q\}$ using $P \Rightarrow P$ and $Q \Rightarrow Q$ (pRef).
- If $M = PQ \to_{\textsc{AppLam}} P'Q = N$, by IH $P \Rightarrow P'$ so $M \Rightarrow N$ by (pApp).
- If $M = PQ \to_{\textsc{AppLam}} PQ' = N$, analogous.
- If $M = \lambda x.P \to_{\textsc{AppLam}} \lambda x.P' = N$, by IH $P \Rightarrow P'$ so $M \Rightarrow N$ by (pLam).
- If $M = \theta \to_{\textsc{AppLam}} \theta'$, let $\theta = \{c_i \mapsto M_i\}_{i=1,\ldots,n}$ with $M_j \to_{\textsc{AppLam}} M_j'$ for some $1 \leq j \leq n$ and $\theta' = \{c_i \mapsto M_i'\}_{i=1,\ldots,n}$ where $M_i' = M_i$ for $i \neq j$, then by IH $M_j \Rightarrow M_j'$ thus $\theta \Rightarrow \theta'$ by (pCB).
- If $M = \{\!|\theta|\!\}.\, M_1 \to_{\textsc{AppLam}} \{\!|\theta'|\!\}.\, M_1 = N$, by IH $\theta \Rightarrow \theta'$ so $M \Rightarrow N$ by (pCase).
- If $M = \{\!|\theta|\!\}.\, M_1 \to_{\textsc{AppLam}} \{\!|\theta|\!\}.\, M_1' = N$, by IH $M_1 \Rightarrow M_1'$ so $M \Rightarrow N$ by (pCase).

Since $\to_{\textsc{AppLam}} \subseteq \Rightarrow$, it follows that $\overset{*}{\to}_{\textsc{AppLam}} \subseteq \overset{*}{\Rightarrow}$.

Now we verify that $\Rightarrow \subseteq \overset{*}{\to}_{\textsc{AppLam}}$ (so $\overset{*}{\Rightarrow} \subseteq \overset{*}{\to}_{\textsc{AppLam}}$ will follow). We prove that $P \overset{*}{\to}_{\textsc{AppLam}} Q$ by induction on the derivation of $P \Rightarrow Q$:

- if (pRef) was applied, trivial
- for (pAppLam), $P = (\lambda x.M)N \to_{\textsc{AppLam}} M\{\mathtt{x} := N\} \overset{*}{\to}_{\textsc{AppLam}} M'\{\mathtt{x} := N'\} = Q$, where the latter derivation uses Corollary 7.6.18, since by IH $M \overset{*}{\to}_{\textsc{AppLam}} M'$ and $N \overset{*}{\to}_{\textsc{AppLam}} N'$
- for (pApp), $P = MN \overset{*}{\to}_{\textsc{AppLam}} M'N' = Q$ since by IH $M \overset{*}{\to}_{\textsc{AppLam}} M'$ and $N \overset{*}{\to}_{\textsc{AppLam}} N'$
- for (pLam), $P = \lambda x.M \overset{*}{\to}_{\textsc{AppLam}} \lambda x.M' = Q$ since by IH $M \overset{*}{\to}_{\textsc{AppLam}} M'$
- for (pCB), $P = \{c_i \mapsto M_i\}_{i=1,\ldots,n} \overset{*}{\to}_{\textsc{AppLam}} \{c_i \mapsto M_i'\}_{i=1,\ldots,n} = Q$ since by IH $M_i \overset{*}{\to}_{\textsc{AppLam}} M_i'$ for $1 \leq i \leq n$
- for (pCase), $P = \{\!|\theta|\!\}.\, M \overset{*}{\to}_{\textsc{AppLam}} \{\!|\theta'|\!\}.\, M' = Q$ since by IH $\theta \overset{*}{\to}_{\textsc{AppLam}} \theta'$ and $M \overset{*}{\to}_{\textsc{AppLam}} M'$

□

Now we get

Corollary 7.6.20 (AppLam and CaseCase commutation) *For all terms and case bindings M, M_1, M_2, if $M \overset{*}{\to}_{\textsc{AppLam}} M_1$ and $M \overset{*}{\to}_{\textsc{CaseCase}} M_2$, then there exists M_3 such that*

$M_1 \xrightarrow{*}_{\text{CASECASE}} M_3$ *and* $M_2 \xrightarrow{*}_{\text{APPLAM}} M_3$*. In other words, the following diagram holds:*

$$\begin{array}{ccc} M & \xrightarrow[CC]{} & M_2 \\ {\scriptstyle AL}\downarrow & & \vdots\,{\scriptstyle AL} \\ M_1 & \xrightarrow[CC]{} & M_3 \end{array}$$

PROOF: Using Commutation Lemma and Lemma 7.6.17 we get that $\Rightarrow$ commutes with $\rightarrow_{\text{CASECASE}}$, therefore by Lemma 7.6.19 we conclude. □

7.6.5 Commutation of AL with AL

We can extend the confluence of β-reduction on Λ to the confluence of APPLAM on $\Lambda_{\mathcal{C}} + \mathcal{B}$.

Lemma 7.6.21 $\Rightarrow$ *satisfies the diamond property*

PROOF: We prove that the following diagrams hold (respectively for terms and case bindings):

$$\begin{array}{ccc} M & \Longrightarrow & M_2 \\ \Downarrow & & \vdots \\ M_1 & \dashrightarrow & M_3 \end{array} \qquad \begin{array}{ccc} \theta & \Longrightarrow & \theta_2 \\ \Downarrow & & \vdots \\ \theta_1 & \dashrightarrow & \theta_3 \end{array}$$

by induction on the derivation of $M \Rightarrow M_1$ (respectively $\theta \Rightarrow \theta_1$). We have the following cases:

1. (pRef) was applied, with $M = M_1$, take $M_3 = M_2$.
2. (pAppLam) was applied, with $M = (\lambda x.P)Q$, $M_1 = P'\{\mathtt{x} := Q'\}$ with $P \Rightarrow P'$, $Q \Rightarrow Q'$, so by Proposition 7.6.13 (5) and (6) we have that
   - either $M_2 = (\lambda x.P'')Q''$ with $P \Rightarrow P''$, $Q \Rightarrow Q''$, in which case

$$\begin{array}{ccc} (\lambda x.P)Q & \Longrightarrow & (\lambda x.P'')Q'' \\ \Downarrow & & \\ P'\{\mathtt{x} := Q'\} & & \end{array}$$

   By IH the following diagrams close for some P''', Q''':

$$\begin{array}{ccc} P & \Longrightarrow & P'' \\ \Downarrow & & \vdots \\ P' & \dashrightarrow & P''' \end{array} \qquad \begin{array}{ccc} Q & \Longrightarrow & Q'' \\ \Downarrow & & \vdots \\ Q' & \dashrightarrow & Q''' \end{array}$$

   By Lemma 7.6.15, the original diagram is closed taking $M_3 = P'''\{\mathtt{x} := Q'''\}$.

- or $M_2 = P''\{\mathtt{x} := Q''\}$ with $P \Rightarrow P''$, $Q \Rightarrow Q''$, in which case

$$\begin{array}{ccc} (\lambda x.P)Q & \Longrightarrow & P''\{\mathtt{x} := Q''\} \\ \Downarrow & & \\ P'\{\mathtt{x} := Q'\} & & \end{array}$$

By IH the following two diagrams close for some P''', Q''':

$$\begin{array}{ccc} P & \Longrightarrow & P'' \\ \Downarrow & & \vdots \\ P' & \dashrightarrow & P''' \end{array} \qquad \begin{array}{ccc} Q & \Longrightarrow & Q'' \\ \Downarrow & & \vdots \\ Q' & \dashrightarrow & Q''' \end{array}$$

By Lemma 7.6.15, the original diagram is closed taking $M_3 = P'''\{\mathtt{x} := Q'''\}$.

3. (pLam) was applied, with $M = \lambda x.P$, $M_1 = \lambda x.P'$, $P \Rightarrow P''$, and by Proposition 7.6.13 (5) $M_2 = \lambda x.P'$ with $P \Rightarrow P'$, in which case

$$\begin{array}{ccc} \lambda x.P & \Longrightarrow & \lambda x.P' \\ \Downarrow & & \\ \lambda x.P'' & & \end{array}$$

By IH we have

$$\begin{array}{ccc} P & \Longrightarrow & P' \\ \Downarrow & & \vdots \\ P'' & \dashrightarrow & P''' \end{array} \quad \textit{then} \quad \begin{array}{ccc} & & \lambda x.P' \\ & & \vdots \\ \lambda x.P'' & \dashrightarrow & \lambda x.P''' \end{array}$$

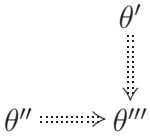
4. (pCB) was applied, with $\theta = \{c_i \mapsto N_i\}_{i=1,\ldots,n}$, $\theta'' = \{c_i \mapsto N_i''\}_{i=1,\ldots,n}$ with $N_i \Rightarrow N_i''$ for $1 \leq i \leq n$ and by Proposition 7.6.13 (7) $\theta' = \{c_i \mapsto N_i'\}_{i=1,\ldots,n}$ with $N_i \Rightarrow N_i'$ for $1 \leq i \leq n$, in which case

$$\begin{array}{ccc} \theta & \Longrightarrow & \theta' \\ \Downarrow & & \\ \theta'' & & \end{array}$$

so for every $1 \leq i \leq n$ we have by IH the diagram

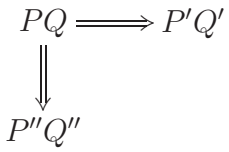
$$\begin{array}{ccc} N_i & \Longrightarrow & N_i' \\ \Downarrow & & \vdots \\ N_i'' & \dashrightarrow & N_i''' \end{array}$$

then taking $\theta''' = \{c_i \mapsto N_i'''\}_{i=1,\ldots,n}$

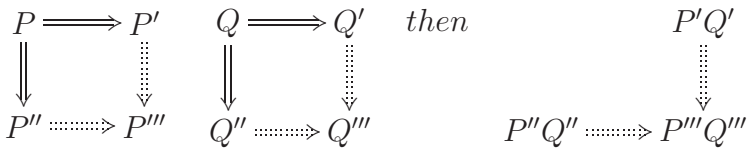


5. (pApp) was applied, with $M = PQ$, $M_1 = P''Q''$ with $P \Rightarrow P''$, $Q \Rightarrow Q''$, so by Proposition 7.6.13 (6) we have that

   - either $M_2 = P'Q'$ with $P \Rightarrow P'$, $Q \Rightarrow Q'$, in which case

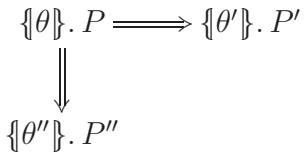
     

     and by IH we have the diagrams

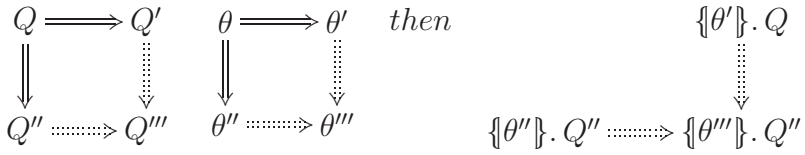
     

   - or $M_2 = N'\{\mathtt{x} := Q'\}$ with $P = \lambda x.N$, $N \Rightarrow N'$ and $Q \Rightarrow Q'$, in which case by Proposition 7.6.13 (5) $P'' = \lambda x.N''$ with $N \Rightarrow N''$. This case is symmetrical with the first item of case 2, so the diagram is closed analogously.

6. (pCase) was applied, with $M = \{\!|\theta|\!\}.\, Q$, $M_1 = \{\!|\theta''|\!\}.\, Q''$, so by Proposition 7.6.13 (8) we have that $M_2 = \{\!|\theta'|\!\}.\, P'$ with $\theta \Rightarrow \theta'$, $P \Rightarrow P'$ and

   and by IH we have

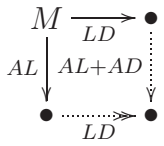
□

Corollary 7.6.22 (Confluence of AppLam) AppLam *is confluent.*

Proof: By previous lemma $\Rightarrow$ satisfies the diamond property and hece it is confluent. We conclude by Lemma 7.6.19 that AppLam is confluent. □

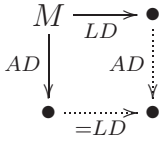
7.6.6 Commutation of $AL + AD$ with LD

Lemma 7.6.23 *The following diagram holds for terms and case bindings:*



Proof: By induction on M. There is only one critical pair, which closes (Figure 7.3). It uses Corollary 7.6.3 (1) and (2). Note that AppLam may erase or duplicate redexes, thus the $\xrightarrow{*}$ arrow. □

Lemma 7.6.24 *The following diagram holds for terms and case bindings:*



Proof: By induction on M. There are no critical pairs, thus it closes. Note that AppDai may erase redexes, thus the $=$ subscript. □

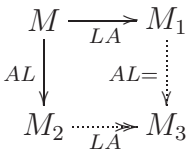
Lemma 7.6.25 AppLam + AppDai//LamDai

Proof: By induction on M. Use lemmas 7.6.23 and 7.6.24 to conclude that AppLam+AppDai strongly commutes with LamDai, therefore they commute. □

7.6.7 Commutation of AL with LA

Just as in the classical λ-calculus, β (here AppLam) commutes with η (here LamApp). To prove strong commutation between both rules we will use Lemma 7.6.2 (1) and (2) for LamApp.

Lemma 7.6.26 (AppLam and LamApp strong commutation) *For all terms and case bindings M, M_1, M_2, if $M \to_{\text{LamApp}} M_1$ and $M \to_{\text{AppLam}} M_2$, then there exists M_3 such that $M_1 \to_{\text{AppLam}^=} M_3$ and $M_2 \xrightarrow{*}_{\text{LamApp}} M_3$. In other words, the following diagram holds:*



Proof: By induction on M.

- If $M = x, c, \maltese$ it holds vacuously (no redexes).

- for M an application and when the AppLam-redex occurs at the root, we have the cases:

  1. internal LamApp at the left

$$\begin{array}{ccc} (\lambda x.P)Q & \xrightarrow[LA]{} & (\lambda x.P')Q \\ {\scriptstyle AL}\downarrow & & \\ P\{\mathtt{x} := Q\} & & \end{array}$$

     with $P \to_{\text{LamApp}} P'$, so by Lemma 7.6.2 (1),
     $P\{\mathtt{x} := Q\} \to_{\text{LamApp}} P'\{\mathtt{x} := Q\}$ thus

$$\begin{array}{ccc} & & (\lambda x.P')Q \\ & & {\scriptstyle AL}\vdots \\ P\{\mathtt{x} := Q\} & \overset{+}{\underset{LA}{\cdots\!\!\to}} & P'\{\mathtt{x} := Q\} \end{array}$$

     and the diagram is closed

  2. internal LamApp at the right

$$\begin{array}{ccc} (\lambda x.P)Q & \xrightarrow[LA]{} & (\lambda x.P)Q' \\ {\scriptstyle AL}\downarrow & & \\ P\{\mathtt{x} := Q\} & & \end{array}$$

     with $Q \to_{\text{LamApp}} Q'$, so by Lemma 7.6.2 (2),
     $P\{\mathtt{x} := Q\} \overset{*}{\to}_{\text{LamApp}} P\{\mathtt{x} := Q'\}$ thus

$$\begin{array}{ccc} & & (\lambda x.P)Q' \\ & & {\scriptstyle AL}\vdots \\ P\{\mathtt{x} := Q\} & \overset{+}{\underset{LA}{\cdots\!\!\to}} & P\{\mathtt{x} := Q'\} \end{array}$$

     and the diagram is closed

  3. the AppLam-redex overlaps with the LamApp-redex, i.e.

$$\begin{array}{ccc} (\lambda x.Px)Q & \xrightarrow[LA]{} & PQ \\ {\scriptstyle AL}\downarrow & & \\ (Px)\{\mathtt{x} := Q\} & & \end{array}$$

     with $x \notin FV(P)$, but then $(Px)\{\mathtt{x} := Q\} = P\{\mathtt{x} := Q\}x\{\mathtt{x} := Q\} = PQ$ so the diagram is closed (in 0 steps).

- For $M = PQ$ and both AppLam and LamApp-reductions internal to P

$$\begin{array}{ccc} PQ & \xrightarrow[LA]{} & P_1Q \\ {\scriptstyle AL}\downarrow & & \\ P_2Q & & \end{array}$$

  by IH there exists P_3 such that

$$\begin{array}{ccc} P & \xrightarrow[LA]{} & P_1 \\ {\scriptstyle AL}\downarrow & & \vdots\,{\scriptstyle AL=} \\ P_2 & \xrightarrow[LA]{} & P_3 \end{array}$$

  thus taking $M_3 = P_3Q$ the diagram is closed

- For $M = PQ$ and both AppLam and LamApp-reductions internal to Q

$$\begin{array}{ccc} PQ & \xrightarrow[LA]{} & PQ_1 \\ {\scriptstyle AL}\downarrow & & \\ PQ_2 & & \end{array}$$

  it is analogous to the previous case

- For $M = PQ$ and one of the reductions occurs in P and the other in Q, the diagram is one of the following two:

$$\begin{array}{ccc} PQ & \xrightarrow[LA]{} & PQ_1 \\ {\scriptstyle AL}\downarrow & & \\ P_1Q & & \end{array} \qquad \begin{array}{ccc} PQ & \xrightarrow[LA]{} & P_1Q \\ {\scriptstyle AL}\downarrow & & \\ PQ_1 & & \end{array}$$

  and both close to P_1Q_1 in one step

- For $M = \lambda x.P$ and the LamApp-redex is internal, we have

$$\begin{array}{ccc} \lambda x.P & \xrightarrow[LA]{} & \lambda x.P_1 \\ {\scriptstyle AL}\downarrow & & \\ \lambda x.P_2 & & \end{array}$$

  so by IH there exists P_3 such that

$$\begin{array}{ccc} P & \xrightarrow[LA]{} & P_1 \\ {\scriptstyle AL}\downarrow & & \vdots\,{\scriptstyle AL=} \\ P_2 & \xrightarrow[LA]{} & P_3 \end{array}$$

  thus taking $M_3 = \lambda x.P_3$ the diagram is closed

- For $M = \lambda x.P$ and the LamApp reduction at the root, we have two cases according to the occurrence of the AppLam-redex:

$$\begin{array}{ccc} \lambda x.Px & \xrightarrow[LA]{} & P \\ {\scriptstyle AL}\downarrow & & \downarrow{\scriptstyle AL} \\ \lambda x.P_2x & \xrightarrow[LA]{} & P_2 \end{array} \qquad \begin{array}{ccc} \lambda x.(\lambda y.M)x & \xrightarrow[LA]{} & \lambda y.M \\ {\scriptstyle AL}\downarrow & \overset{\alpha}{\nearrow} & \\ \lambda x.M\{\mathtt{y} := x\} & & \end{array}$$

since $x \notin FV(P)$, $x \notin FV(P_2)$, $x \notin FV(M)$, $x \notin FV(\lambda y.M)$.

- For $M = \theta = \{c_i \mapsto N_i\}_{i=1,\ldots,n}$, with $M_1 = \theta' = \{c_i \mapsto N_i'\}$, $M_2 = \theta'' = \{c_i \mapsto N_i''\}$, then

  1. there is a j such that both redexes occur in N_j, then by IH

$$\begin{array}{ccc} N_j & \xrightarrow[LA]{} & N_j' \\ {\scriptstyle AL}\downarrow & & \vdots{\scriptstyle AL=} \\ N_j'' & \xrightarrow[LA]{} & N_j''' \end{array}$$

     so take $\theta''' = \{c_i \mapsto N_i'''\}_{i=1,\ldots,n}$ where $N_i''' = N_i$ for $i \neq j$ and the diagram is closed

  2. the AppLam-redex occurs in N_j with reduct N_j'' and the LamApp-redex occurs in N_k with reduct N_k', then take $\theta''' = \{c_i \mapsto N_i'''\}_{i=1,\ldots,n}$ where $N_i''' = N_i$ for $i \neq j, k$, $N_j''' = N_j''$, $N_k''' = N_k'$ and the diagram is closed

- For $M = \{\!\{\theta\}\!\}.\, P$ and

$$\begin{array}{ccc} \{\!\{\theta\}\!\}.\, P & \xrightarrow[LA]{} & \{\!\{\theta_1\}\!\}.\, P \\ {\scriptstyle AL}\downarrow & & \\ \{\!\{\theta_2\}\!\}.\, P & & \end{array}$$

  by IH there exists θ_3 such that

$$\begin{array}{ccc} \theta & \xrightarrow[LA]{} & \theta_1 \\ {\scriptstyle AL}\downarrow & & \vdots{\scriptstyle AL=} \\ \theta_2 & \xrightarrow[LA]{} & \theta_3 \end{array}$$

  thus taking $M_3 = \{\!\{\theta_3\}\!\}.\, P$ the diagram is closed

- For $M = \{\!\{\theta\}\!\}.\, P$ and

$$\begin{array}{ccc} \{\!\{\theta\}\!\}.\, P & \xrightarrow[LA]{} & \{\!\{\theta\}\!\}.\, P_1 \\ {\scriptstyle AL}\downarrow & & \\ \{\!\{\theta\}\!\}.\, P_2 & & \end{array}$$

  it is analogous to the previous case

- For $M = \{\!\{\theta\}\!\}.\, P$ with the AppLam-redex in P and the LamApp-redex in θ, take $M_3 = \{\!\{\theta_1\}\!\}.\, P_1$ and the next diagram holds:

$$\begin{array}{ccc} \{\!\{\theta\}\!\}.\, P & \xrightarrow[LA]{} & \{\!\{\theta_1\}\!\}.\, P \\ {\scriptstyle AL}\downarrow & & \downarrow{\scriptstyle AL} \\ \{\!\{\theta\}\!\}.\, P_1 & \xrightarrow[LA]{} & \{\!\{\theta_1\}\!\}.\, P_1 \end{array}$$

- For $M = \{\!\{\theta\}\!\}.\, P$ with the AppLam-redex in θ and the LamApp-redex in P, analogously.

□

Now we get

Corollary 7.6.27 (AppLam and LamApp commutation) *For all terms and case bindings M, M_1, M_2, if $M \xrightarrow{*}_{\text{LamApp}} M_1$ and $M \xrightarrow{*}_{\text{AppLam}} M_2$, then there exists M_3 such that $M_1 \xrightarrow{*}_{\text{AppLam}} M_3$ and $M_2 \xrightarrow{*}_{\text{LamApp}} M_3$. In other words, the following diagram holds:*

$$\begin{array}{ccc} M & \xrightarrow[LA]{} & M_1 \\ {\scriptstyle AL}\downarrow & & \downarrow{\scriptstyle AL} \\ M_2 & \xrightarrow[LA]{} & M_3 \end{array}$$

Proof: By the previous lemma they strongly commute, therefore by the Commutation Lemma they commute. □

7.6.8 Commutation of $AL + AD$ with $LA + LD$

Lemma 7.6.28 *The following diagram holds for terms and case bindings:*

$$\begin{array}{ccc} M & \xrightarrow[LA]{} & M_1 \\ {\scriptstyle AD}\downarrow & & \downarrow{\scriptstyle =AD} \\ M_2 & \xrightarrow[=LA+LD]{} & M_3 \end{array}$$

Proof: By induction on M. Note that AppDai may eliminate redexes, thus the $\to^=$ derivation. □

Lemma 7.6.29 *The following diagram holds for terms and case bindings:*

$$\begin{array}{ccc} M & \xrightarrow[LA+LD]{} & M_1 \\ {\scriptstyle AL+AD}\downarrow & & \downarrow{\scriptstyle =AL+AD} \\ M_2 & \xrightarrow[LA+LD]{} & M_3 \end{array}$$

Proof: By induction on M, using lemmas 7.6.23, 7.6.24, 7.6.28 and 7.6.26. □

Lemma 7.6.30 AppLam + AppDai//LamApp + LamDai

Proof: Using Lemma 7.6.29, they strongly commute, therefore by the Commutation Lemma they commute. □

7.6.9 Commutation of $AL + CL$ with $LA + CA$

We begin with auxiliary lemmas.

Lemma 7.6.31 *The following diagram holds for terms and case bindings:*

$$\begin{array}{ccc} M & \xrightarrow[LA]{} & M_1 \\ {\scriptstyle CL}\downarrow & & \vdots\,{\scriptstyle =CL} \\ M_2 & \xrightarrow[=CA]{} \cdots\xrightarrow[LA]{} & M_3 \end{array}$$

Proof: By induction on M. The interesting cases are the overlaps given by

1. an external CaseLam-reduction as

$$\begin{array}{ccccc} \{\!|\theta|\!\}.\,(\lambda x.Mx) & & \xrightarrow[LA]{} & & \{\!|\theta|\!\}.\,M \\ {\scriptstyle CL}\downarrow & & & & \| \\ \lambda x.(\{\!|\theta|\!\}.\,(Mx)) & \xrightarrow[CA]{} & \lambda x.(\{\!|\theta|\!\}.\,M)x & \xrightarrow[LA]{} & \{\!|\theta|\!\}.\,M \end{array}$$

   where $x \notin FV(\theta)$, $x \notin FV(M)$, thus $x \notin FV(\theta) \cup FV(M) = FV(\{\!|\theta|\!\}.\,M)$ and the diagram holds.

2. an internal CaseLam-reduction $M \to_{\text{CaseLam}} M'$ as

$$\begin{array}{ccc} \lambda x.Mx & \xrightarrow[LA]{} & M \\ {\scriptstyle CL}\downarrow & & \vdots\,{\scriptstyle CL} \\ \lambda x.M'x & \xrightarrow[LA]{} & M' \end{array}$$

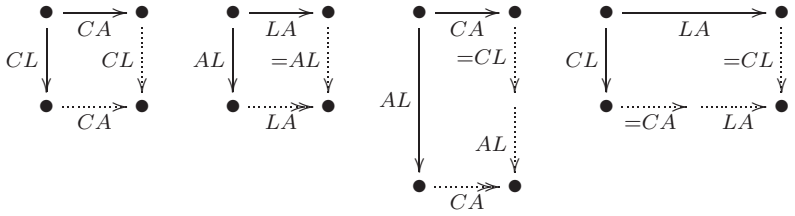
   where $x \notin FV(M)$, and since by Lemma 7.2.1 $FV(M') \subseteq FV(M)$, $x \notin FV(M')$ and the diagram holds.

□

Lemma 7.6.32 *The following diagram holds for terms and case bindings:*

$$\begin{array}{ccc} M & \xrightarrow[LA+CA]{} & M_1 \\ & & \vdots\,{\scriptstyle =CL} \\ {\scriptstyle AL+CL}\downarrow & & \\ & & \vdots\,{\scriptstyle =AL} \\ M_2 & \xrightarrow[LA+CA]{} & M_3 \end{array}$$

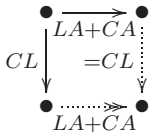
Proof: We reduce the proof to check the four cases given respectively by the following diagrams:



The first diagram holds by Lemma 7.6.8. The second diagram holds by Lemma 7.6.26. The third diagram holds by Lemma 7.6.9. The fourth diagram holds by Lemma 7.6.31. □

Remark 7.6.33 *The diagrams in the proof of Lemma 7.6.32 show that*

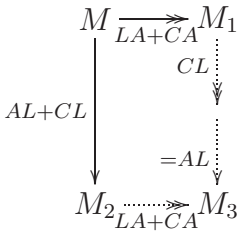
1. LamApp+CaseApp *strongly commutes with* CaseLam *(and hence they commute, by the Commutation Lemma), since the following diagram is a direct consequence of the* 1^{st} *and* 4^{th} *diagrams above:*



2. CaseApp *strongly commutes with* CaseLam.

The first item above will be used in the next lemma:

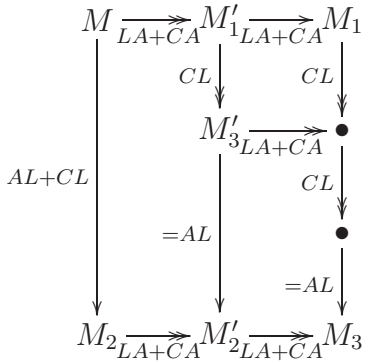
Lemma 7.6.34 *The following diagram holds for terms and case bindings:*



Proof: The proof is done by induction on the (LamApp+CaseApp)* derivation, using Lemma 7.6.32, Remark 7.6.33(1) and the Commutation Lemma. We distinguish the following cases:

- If the derivation $M \xrightarrow{*}_{LA+CA} M_1$ has 0 steps, the result is obvious.
- If it has 1 step, it is Lemma 7.6.32.
- So let us assume it has ≥ 2 steps. We proceed by lexicographic induction on the pair

($LA+CA+CL$-depth of M, length of the $LA+CA$-derivation). The picture is



The left rectangle is closed by IH since the second component of the pair decreases. The upper right square can be closed by Remark 7.6.33. At M'_3 the lexicographic pair is clearly less than the value at M, since the length of $M \xrightarrow{*}_{LA+CA} M'_1$ is ≥ 1, so the IH allows to close the lower right rectangle.

□

So finally we get

Lemma 7.6.35 AppLam + CaseLam//LamApp + CaseApp

Proof: Consequence of the previous lemma, using induction on the length of the AppLam + CaseLam-derivation. □

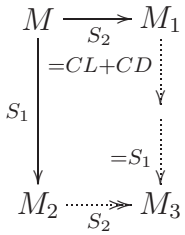
7.6.10 Commutation of $AL+AD+CL+CD$ with $LA+LD+CA+CD$

For simplicity, let us call

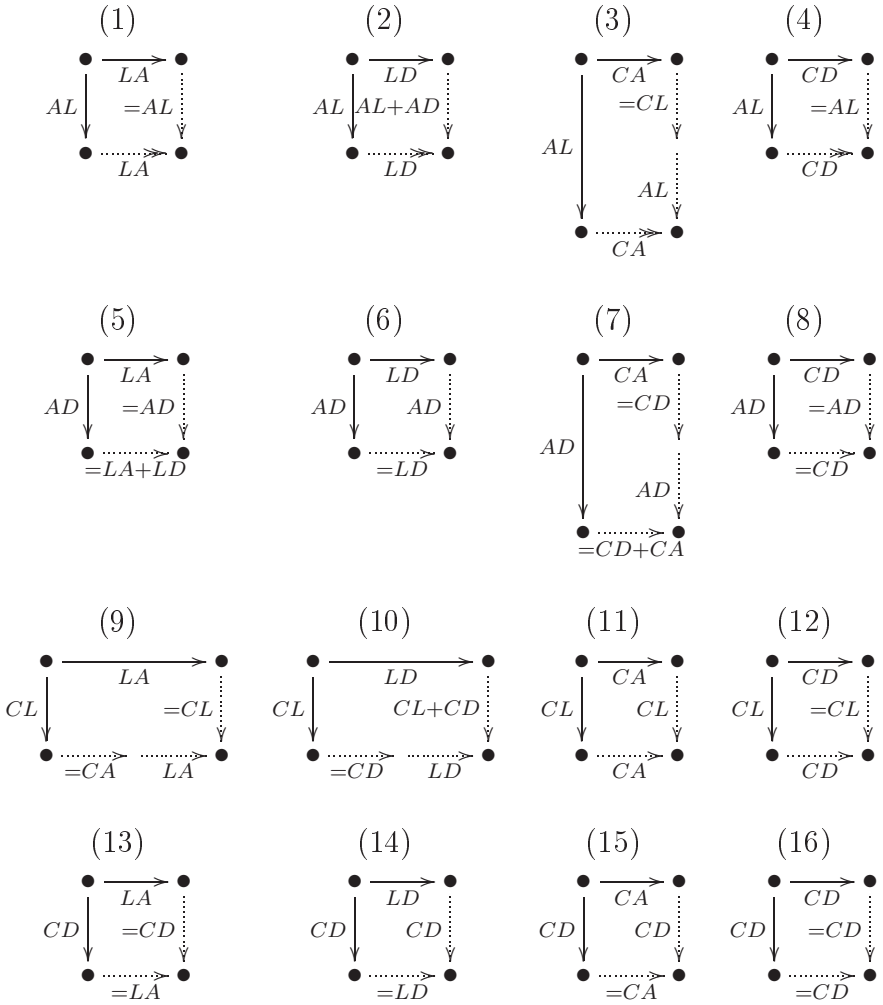
- S_1 = AppLam + AppDai + CaseLam + CaseDai
- S_2 = LamApp + LamDai + CaseApp + CaseDai.

Then we can give the

Lemma 7.6.36 *The following diagram holds for terms and case bindings:*



PROOF: We reduce the proof to check the different cases. In this proof, when there is no critical pair, the diagram can be closed with a single step, noting that certain reduction steps can eventually erase the other redex (thus the presence of "=" subscripts in those cases), and also an APPLAM-reduction step may erase or duplicate the other redex (thus the presence of a $\xrightarrow{*}$ arrow). For the cases where there is a critical pair, we refer the reader to Figures 7.2 and 7.3. We have sixteen cases given respectively by the following diagrams:



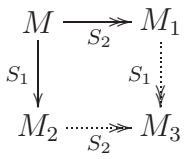
Remarks for the preceding diagrams:

1. use Lemma 7.6.26 to close, note that AL may erase or duplicate the LA-redex
2. one critical pair, AL may erase or duplicate the LD-redex
3. one critical pair, AL may erase or duplicate the CA-redex
4. no critical pair, AL may erase or duplicate the CD-redex, CD may erase the AL-redex
5. one critical pair, AD may erase the LA-redex and LA may erase or duplicate the AD-redex
6. no critical pair, AD may erase the LD-redex

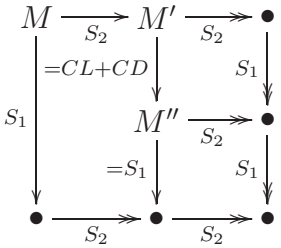
7. one critical pair, AD may erase the CA-redex
8. no critical pair, AD may erase the CD-redex and CD may erase the AD-redex
9. one critical pair, LA may erase the CL-redex
10. one critical pair, these rules do not erase nor duplicate redexes
11. no critical pair, these rules do not erase nor duplicate redexes
12. no critical pair, CD may erase the CL-redex
13. no critical pair, LA may erase the CD-redex and CD may erase the LA-redex
14. no critical pair, CD may erase the LD-redex
15. no critical pair, CD may erase the CA-redex
16. no critical pair, one CD may erase the other CD-redex

□

Lemma 7.6.37 *The following diagram holds for terms and case bindings:*



PROOF: By well-founded induction on the $S_2 + \text{CASELAM} + \text{CASEDAI}$-depth of M. If the length of the S_2-derivation is 0, trivial. If it is 1, use Lemma 7.6.36. Else, the picture is:



where the left rectangle can be closed by Lemma 7.6.36, and the top right and lower right squares can be closed by IH since we have that $depth(M') < depth(M)$ and $depth(M'') < depth(M)$. □

Lemma 7.6.38 $S_1 // S_2$

PROOF: By Lemma 7.6.37 and induction on the S_1-derivation. □

7.6.11 General Commutation and Confluence

As a consequence of the previous lemmata we have the full version of theorem 7.5.4. It states that all pairs of subsystems of $\lambda\mathcal{B}_{\mathcal{C}}$ which satisfy the BCC commute.

Theorem 7.6.39 (General commutation) — *Given two subsystems s_1 and s_2, the following propositions are equivalent:*

1. *The pair (s_1, s_2) fulfills the binary closure conditions;*
2. *The reduction relations s_1 and s_2 weakly commute;*
3. *The reduction relations s_1 and s_2 commute.*

PROOF: (3)⇒(2) is obvious. (1)⇒(2) and (2)⇒(1) are Theorem 7.5.4. (2)⇒(3) is a consequence of database results and the inferences made by the program. As a sample we include the confluence proof for the entire 9-rule system:

1. $(AL \vdash CR)$ by a database lemma (Corollary 7.6.22)
2. $(AL \;//\; AD)$ by a database lemma (Lemma 7.6.4)
3. $(AL \;//\; CO)$ by a database lemma (Lemma 7.6.5)
4. $(AL \;//\; CD)$ by a database lemma (Lemma 7.6.6)
5. $(AL \;//\; CL)$ by a database lemma (Lemma 7.6.7)
6. $(AL \;//\; CD + CL)$ since $(CD \;//\; AL)$ and $(CL \;//\; AL)$
7. $(AL \;//\; AD + CD + CL)$ since $(AD \;//\; AL)$ and $(CD + CL \;//\; AL)$
8. $(AL \;//\; AL + AD + CD + CL)$ since $(AL \vdash CR)$ and $(AD + CD + CL \;//\; AL)$
9. $(AL \;//\; CC)$ by a database lemma (Corollary 7.6.20)
10. $(LA + LD + CD + CA \;//\; AL + AD + CD + CL)$ by a database lemma (Lemma 7.6.38)
11. $(AL \;//\; CL + CC)$ since $(CL \;//\; AL)$ and $(CC \;//\; AL)$
12. $(AL \;//\; CD + CL + CC)$ since $(CD \;//\; AL)$ and $(CL + CC \;//\; AL)$
13. $(AL \;//\; CO + CD + CL + CC)$ since $(CO \;//\; AL)$ and $(CD + CL + CC \;//\; AL)$
14. $(AL \;//\; AD + CO + CD + CL + CC)$ since $(AD \;//\; AL)$ and $(CO + CD + CL + CC \;//\; AL)$
15. $(AD + CD + CL \;//_w\; AD + CO + CD + CL + CC)$ since $(AD + CD + CL, AD + CO + CD + CL + CC) \vdash BCC$
16. $(AD + CD + CL \;//\; AD + CO + CD + CL + CC)$ since $(AD + CD + CL \;//_w\; AD + CO + CD + CL + CC)$ and $(AD + CD + CL + AD + CO + CD + CL + CC \vdash SN)$

17. $(AL+AD+CD+CL \;//\; AD+CO+CD+CL+CC)$ since $(AL \;//\; AD+CO+CD+CL+CC)$ and $(AD+CD+CL \;//\; AD+CO+CD+CL+CC)$
18. $(AL+AD+CD+CL \;//\; AD+LA+LD+CO+CD+CA+CL+CC)$ since $(LA+LD+CD+CA \;//\; AL+AD+CD+CL)$ and $(AD+CO+CD+CL+CC \;//\; AL+AD+CD+CL)$
19. $(AL+AD+CD+CL \;//\; AL+AD+LA+LD+CO+CD+CA+CL+CC)$ since $(AL \;//\; AL+AD+CD+CL)$ and $(AD+LA+LD+CO+CD+CA+CL+CC \;//\; AL+AD+CD+CL)$
20. $(LA+LD+CO+CD+CA+CL+CC \;//_w\; AD+LA+LD+CO+CD+CA+CL+CC)$ since $(LA+LD+CO+CD+CA+CL+CC, AD+LA+LD+CO+CD+CA+CL+CC) \vdash BCC$
21. $(LA+LD+CO+CD+CA+CL+CC \;//\; AD+LA+LD+CO+CD+CA+CL+CC)$ since $(LA+LD+CO+CD+CA+CL+CC \;//_w\; AD+LA+LD+CO+CD+CA+CL+CC)$ and $(LA+LD+CO+CD+CA+CL+CC + AD(+LA+LD+CO+CD+CA+CL+CC) \vdash SN)$
22. $(AD+LA+LD+CO+CD+CA+CL+CC \;//\; AL+AD+LA+LD+CO+CD+CA+CL+CC)$ since $(AL+AD+CD+CL \;//\; AD+LA+LD+CO+CD+CA+CL+CC)$ and $(LA+LD+CO+CD+CA+CL+CC \;//\; AD+LA+LD+CO+CD+CA+CL+CC)$
23. $(AL+AD+LA+LD+CO+CD+CA+CL+CC \vdash CR)$ since $(AL+AD+CD+CL \;//\; AL+AD+LA+LD+CO+CD+CA+CL+CC)$ and $(AD+LA+LD+CO+CD+CA+CL+CC \;//\; AL+AD+LA+LD+CO+CD+CA+CL+CC)$

□

As a consequence of Theorem 7.6.39 we have the confluence of all the subsystems of $\lambda\mathcal{B}_\mathcal{C}$ which satisfy the CC, including itself.

It is interesting that not all the lemmas in the database are used in this proof, but only 7 of them: 7.6.22 $(AL//AL)$, 7.6.4 $(AL//AD)$, 7.6.5 $(AL//CO)$, 7.6.20 $(AL//CC)$, 7.6.6 $(AL//CD)$, 7.6.7 $(AL//CL)$ and 7.6.38 $(LA+LD+CD+CA//AL+AD+CD+CL)$. This happens because the preceding 23-step proof is a shortest proof (minimum number of steps) for the confluence of the whole system. For other subsystems other lemmas in the database are needed. We are currently investigating the existence of alternative proofs which may become preferable with given criteria such as using a minimum number of lemmas in the database. To give a single example, there is another 52-step proof of confluence of $\lambda\mathcal{B}_\mathcal{C}$ using the same database lemmas as above plus Lemma 7.6.11 (CaseApp//AppLam + CaseLam).

Corollary 7.6.40 (General confluence) — *Given a subsystem s, the following propositions are equivalent:*

1. *The subsystem s fulfills the closure conditions;*

Counting every ordered pair of systems (s_1, s_2):

Rules:	9	
Subsystems:	512	$(= 2^9$ including $\lambda\mathcal{B}_{\mathcal{C}}$ and $\emptyset)$
Pairs of subsystems:	262144	$(= 512^2)$
SN subsystems:	256	(including $\mathcal{B}_{\mathcal{C}}$ and $\emptyset$)
Commuting pairs:	26544	(the same are weakly commuting)
CR subsystems:	248	(the same are weakly confluent)

Counting (s_1, s_2) and (s_2, s_1) as a single pair of systems and excluding $\emptyset$:

	Pairs(s_1, s_2)	$s_1 = s_2$
Subsystems	131328	512
SN + commuting (= ¬ AppLam+ BCC) subsystems	5612	160
Weakly commuting (=BCC)	13396	24
Verified by the program	7784	88

Figure 7.5: Statistics for $\lambda\mathcal{B}_{\mathcal{C}}$-subsystems

2. *The s-reduction is locally confluent;*

3. *The s-reduction is confluent.*

Proof: Immediate using Theorem 7.6.39 and Remark 7.5.3. □

Corollary 7.6.41 *$\lambda\mathcal{B}_{\mathcal{C}}$ is conservative over λ-calculus, in the sense that: For all $M, N \in \Lambda$, if $\lambda\mathcal{B}_{\mathcal{C}} \vdash M = N$ then $\lambda \vdash M = N$.*

Proof: Follows from confluence of $\lambda\mathcal{B}_{\mathcal{C}}$ and the fact that λ-calculus is a sub-ARS of $\lambda\mathcal{B}_{\mathcal{C}}$ because AppLam and LamApp are the only reduction rules that may apply to an ordinary λ-term in $\lambda\mathcal{B}_{\mathcal{C}}$. □

Finally, our results for the different subsystems of $\lambda\mathcal{B}_{\mathcal{C}}$ are summarized in Figure 7.5.

7.7 Separation

This section addresses the Separation Theorem for $\lambda\mathcal{B}_{\mathcal{C}}$. Since proofs are omitted, we refer the reader to (8), or to the technical report at http://www.dc.uba.ar/people/materias/reescritura/clam.ps which contains full proofs and details for this section.

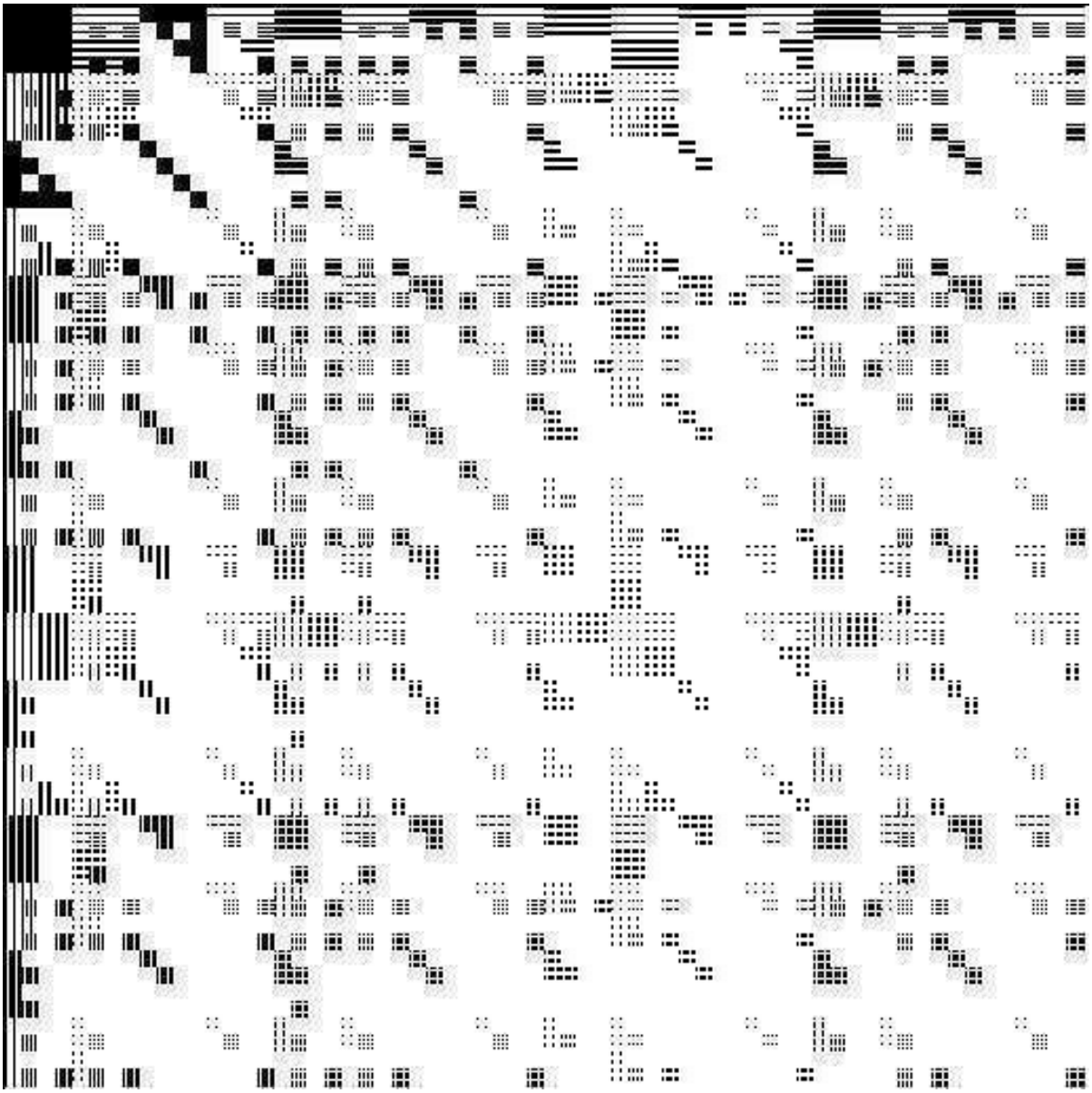


Figure 7.6: Lambda calculus with constructors: subsystem commutation grid

Definition 7.7.1 (Head term) — *We call a* head term *(and write H, H_1, H', etc.) any term that has one of the following four forms:*

Head term $$H \;::=\; x \;\mid\; c \;\mid\; \{\!|\theta|\!\}.\,x \;\mid\; \{\!|\theta|\!\}.\,c \quad (c \notin \mathrm{dom}(\theta))$$

When a head term H is of one of the first three forms (variable, constructor, case binding on a variable), we say that H is defined*. When H is of the last form (case binding on an unbound constructor), we say that H is* undefined*.*

Definition 7.7.2 (Quasi-head normal form) — *A term M is a said to be in* quasi-head normal form *if it has one of the following two forms*

Quasi-hnf $$M \;::=\; \maltese \;\mid\; \lambda x_1 \cdots x_n \,.\, H N_1 \cdots N_k$$

where H is an arbitrary head term, called the head *of M, and where $N_1, \ldots, N_k$ are arbitrary terms.*

Here, the prefix 'quasi' expresses that such terms are in head normal form w.r.t. all the reduction rules, but (possibly) the rule LamApp (a.k.a. η). For instance, the term $\lambda x \,.\, cx$ is in quasi-head normal form according to the definition above, but still contains an η-redex at root position. In what follows, 'quasi' will systematically refer to 'all the reduction rules but LamApp'.

As for head terms, we distinguish *defined* quasi-head normal forms from *undefined* ones, by saying that a quasi-head normal form M is *defined* when it has one of the forms

$$M \;::=\; \maltese \;\mid\; \lambda x_1 \cdots x_n \,.\, H N_1 \cdots N_k \qquad \text{(where } H \text{ is defined)}$$

and that M is *undefined* otherwise, that is, when M has the form

$$M \;::=\; \lambda x_1 \cdots x_n \,.\, (\{\!|\theta|\!\}.\,c) N_1 \cdots N_k \qquad (c \notin \mathrm{dom}(\theta))$$

More generally, we call a *defined* term (resp. an *undefined* term) any term that reduces to a quasi-head normal form which is defined (resp. undefined). The class of defined terms is closed under arbitrary reduction, as for the class of undefined terms. Moreover, the class of undefined terms is closed under arbitrary substitution.

Definition 7.7.3 (Quasi-normal form) — *A term (resp. a case binding) is said to be in* quasi-normal form *when it is in normal form w.r.t. all the reduction rules but* LamApp *(a.k.a. η).*

Terms (resp. case bindings) that are in quasi-normal form are simply called *quasi-normal terms* (resp. *quasi-normal case bindings*). In particular, we call a *quasi-normal head term* any head term H which is in quasi-normal form. These notions have the following syntactic characterization:

Proposition 7.7.4 *Quasi-normal terms, quasi-normal head terms, and quasi-normal case bindings are (mutually) characterized by the following grammar:*

$$\begin{array}{lrcl}
\textbf{\textit{Q.n.-terms}} & N & ::= & \maltese \mid \lambda x_1 \cdots x_n \,.\, H N_1 \cdots N_k \\
\textbf{\textit{Q.n.-head-terms}} & H & ::= & x \mid c \mid \{\!|\theta|\!\}.\, x \mid \{\!|\theta|\!\}.\, c \quad (c \notin \mathrm{dom}(\theta)) \\
\textbf{\textit{Q.n.-case bind.}} & \theta & ::= & c_1 \mapsto N_1; \ldots; c_p \mapsto N_p
\end{array}$$

The notion of *context with one hole* (notation $C[]$, $C'[]$, etc.) is defined in the λ-calculus with constructors as expected. The term obtained by filling a context with one hole $C[]$ with a term M is written $C[M]$, and the composition of two contexts $C[]$ and $C'[]$ is written $C'[C[]]$.

In what follows, we will mainly use contexts of a particular form, namely, *evaluation contexts*:

$$\textbf{Evaluation contexts} \qquad E[] \quad ::= \quad []N_1 \cdots N_n \quad \mid \quad \{\!|\theta|\!\}.\, []N_1 \cdots N_n$$

(The second form should be read $(\{\!|\theta|\!\}.\, [])N_1 \cdots N_n$.)

Notice that the composition $E'[E[]]$ of two evaluation contexts $E[]$ and $E'[]$ is not always an evaluation context, but that it always reduces to an evaluation context using zero, one or several steps of the CaseApp rule, possibly followed by a single step of the CaseCase rule:

$$\begin{array}{rcl}
\left[\; []N_1 \cdots N_k \;\right] N'_1 \cdots N'_{k'} & = & []N_1 \cdots N_k N'_1 \cdots N'_{k'} \\
\left[\; \{\!|\theta|\!\}.\, []N_1 \cdots N_k \;\right] N'_1 \cdots N'_{k'} & = & \{\!|\theta|\!\}.\, []N_1 \cdots N_k N'_1 \cdots N'_{k'} \\
\{\!|\theta'|\!\}.\left[\; []N_1 \cdots N_k \;\right] N'_1 \cdots N'_{k'} & \xrightarrow{*} & \{\!|\theta'|\!\}.\, []N_1 \cdots N_k N'_1 \cdots N'_{k'} \\
\{\!|\theta'|\!\}.\left[\; \{\!|\theta|\!\}.\, []N_1 \cdots N_k \;\right] N'_1 \cdots N'_{k'} & \xrightarrow{*} & \{\!|\theta' \circ \theta|\!\}.\, []N_1 \cdots N_k N'_1 \cdots N'_{k'}
\end{array}$$

Remark 7.7.5 *An evaluation context $E[]$ can always be regarded as a term (of a particular form) that contains exactly one occurrence of a distinguished variable depicted $[]$—the hole. In particular, since the unique occurrence of the hole $[]$ in an evaluation context $E[]$ is outside the scope of all the binders of $E[]$, the operation of replacement $E[M]$ works just as the ordinary operation of substitution $E\{[] := M\}$ of λ-calculus. (This is of course not the case for the general notion of context with one hole—think of $C[x]$ where $C[] = \lambda x \,.\, [].$)*

The daimon $\maltese$ which represents immediate termination naturally "absorbs" all the evaluation contexts:

Lemma 7.7.6 *In any evaluation context $E[]$ one has* $E[\maltese] \xrightarrow{*} \maltese$.

Symmetrically, each subterm of the form $\{\!|\theta|\!\}.\, c$ (with $c \notin \mathrm{dom}(\theta)$) blocks the computation process at head position so that undefined terms "absorb" all evaluation contexts as well:

Lemma 7.7.7 *Given an undefined term U, the term $E[U]$ is undefined in any evaluation context $E[]$.*

The daimon $\maltese$ and undefined terms are thus natural candidates to define the notion of separation:

Definition 7.7.8 (Separability) *We say that two terms M_1 and M_2 are:*

- weakly separable *if there exists a context with one hole $C[]$ such that either:*
  - $C[M_1] \xrightarrow{*} \maltese$ *and $C[M_2]$ is undefined, or*
  - $C[M_2] \xrightarrow{*} \maltese$ *and $C[M_1]$ is undefined;*
- strongly separable *if there exists two contexts $C_1[]$ and $C_2[]$ such that*
  - $C_1[M_1] \xrightarrow{*} \maltese$ *and $C_1[M_2]$ is undefined, and*
  - $C_2[M_2] \xrightarrow{*} \maltese$ *and $C_2[M_1]$ is undefined.*

On the other hand, two undefined terms cannot be separated each other (precisely because their heads block all computations), so that we have to exclude them from our study of the separation property.

Definition 7.7.9 (Completely defined quasi-normal term) *A quasi-normal term M is said to be* completely defined *if it contains no subterm of the form $\{\!|\theta|\!\}.\, c$, where $c \notin \mathrm{dom}(\theta)$.*

In what follows, we will show that distinct completely defined normal terms are weakly separable.

- First we define a syntactic relation between terms, called *disagreement at depth $d \in \mathbb{N}$*, and we show that any pair of distinct normal forms have η-expansions that disagree at some depth (this subsection).
- Then we show (by induction on the depth of disagreement) that any pair of disagreeing quasi-normal terms are weakly separable.

Definition 7.7.10 (Skeleton equivalence) *We say that two defined head terms H_1 and H_2 have* the same skeleton *and write $H_1 \approx H_2$ if either:*

- *$H_1 = x_1$ and $H_2 = x_2$ for some $x_1, x_2 \in \mathcal{V}$, and $x_1 = x_2$; or*
- *$H_1 = c_1$ and $H_2 = c_2$ for some $c_1, c_2 \in \mathcal{C}$, and $c_1 = c_2$; or*
- *$H_1 = \{\!|\theta_1|\!\}.\, x_1$ and $H_2 = \{\!|\theta_2|\!\}.\, x_2$ for some case bindings θ_1, θ_2 and for some $x_1, x_2 \in \mathcal{V}$, and $\mathrm{dom}(\theta_1) = \mathrm{dom}(\theta_2)$ and $x_1 = x_2$.*

Considering the negation of the former equivalence, it is clear that two defined head terms H_1 and H_2 have *not* the same skeleton ($H_1 \not\approx H_2$) when either:

- H_1 is a variable, and H_2 is a constructor (or symmetrically); or
- H_1 is a case-variable, and H_2 is a constructor (or symmetrically); or

- H_1 is a case-variable, and H_2 is a variable (or symmetrically); or
- H_1 and H_2 are both variables, but not the same variable; or
- H_1 and H_2 are both constructors, but not the same constructor; or
- $H_1 = \{\!|\theta_1|\!\}\,.\,x_1$ and $H_2 = \{\!|\theta_2|\!\}\,.\,x_2$ for some case bindings θ_1, θ_2 and for some $x_1, x_2 \in \mathcal{V}$, and either $x_1 \neq x_2$ or $\mathrm{dom}(\theta_1) \neq \mathrm{dom}(\theta_2)$.

(Notice that we do not consider the case of a head term of the form $\{\!|\theta|\!\}\,.\,c$ where $c \notin \mathrm{dom}(\theta)$, which is excluded from our definition.)

Definition 7.7.11 (Disagreement at depth d) *For each numeral $d \in \mathbb{N}$, we define a binary relation on the class of completely defined quasi-normal terms, called the* disagreement relation at depth d. *This relation, written $\mathrm{dis}_d(M_1, M_2)$ ('M_1 and M_2 disagree at depth d'), is defined by induction on $d \in \mathbb{N}$ as follows:*

- *(Base case) We write $\mathrm{dis}_0(M_1, M_2)$ if either:*
  - *$M_1 = \maltese$ and $M_2 = \lambda x_1 \cdots x_n\,.\,H N_1 \cdots N_k$; or*
  - *$M_1 = \lambda x_1 \cdots x_n\,.\,H N_1 \cdots N_k$ and $M_2 = \maltese$; or*
  - *$M_1 = \lambda x_1 \cdots x_n\,.\,H_1 N_{1,1} \cdots N_{1,k_1}$ and $M_2 = \lambda x_1 \cdots x_n\,.\,H_2 N_{2,1} \cdots N_{2,k_2}$ and $H_1 \not\approx H_2$.*
- *(Inductive case) For all $d \in \mathbb{N}$, we write $\mathrm{dis}_{d+1}(M_1, M_2)$ if $M_1 = \lambda x_1 \cdots x_n\,.\,H_1 N_{1,1} \cdots N_{1,k_1}$ and $M_2 = \lambda x_1 \cdots x_n\,.\,H_2 N_{2,1} \cdots N_{2,k_2}$ and $H_1 \approx H_2$, and if either*
  - *$H_1 = \{\!|\theta_1|\!\}\,.\,y$ and $H_2 = \{\!|\theta_2|\!\}\,.\,y$ for some case bindings θ_1, θ_2 and for some variable y, and there is a constructor $c \in \mathrm{dom}(\theta_1) = \mathrm{dom}(\theta_2)$ such that $\mathrm{dis}_d(\theta_1(c), \theta_2(c))$; or*
  - *There is a position $1 \leq k \leq \min(k_1, k_2)$ such that $\mathrm{dis}_d(N_{1,k}, N_{2,k})$.*

Lemma 7.7.12 (Cooking lemma) *If M_1 and M_2 are two completely defined normal terms (w.r.t. all the reduction rules including $\textsc{LamApp} = \eta$) such that $M_1 \neq M_2$, then there exist two completely defined quasi-normal terms M_1' and M_2' such that $M_1' \xrightarrow{*}_\eta M_1$, $M_2' \xrightarrow{*}_\eta M_2$, and $\mathrm{dis}_d(M_1', M_2')$ for some $d \in \mathbb{N}$.*

Theorem 7.7.13 (Separation) *Let M_1 and M_2 be completely defined terms in normal form. If $M_1 \not\equiv M_2$, then M_1 and M_2 are weakly separable.*

7.8 Conclusion and future work

We have introduced a λ-calculus with constructors, $\lambda\mathcal{B}_{\mathcal{C}}$, which models pattern matching with a minimal set of constructs and rewrite rules. We used a set of constants as case binders, which work as substitution-like constructs, in order to perform pattern matching. This matching facility is modeled with adequate rewrite rules. We have extensively studied commutation properties for these rules, and, for the calculus obtained by omitting the β-like and η-like rules (which constitute a case-binder calculus instead of a substitution calculus), we have proved strong normalization. Even with the minimal language we introduced, the confluence and separation results we obtained are somewhat involved.

In order to prove confluence for the whole calculus we started by studying commutation of pairs of one-rule subsystems and designed a program to infer commutation of other subsystems. The "inference rules" of the program are motivated by commutation lemmas. The program guided us by "suggesting" the commutation lemmas which were required to complete the confluence proof sequentially. Once we proved these lemmas, the program combined them with its own inferences and traced the full sequence of inferences for any specific pair of commuting sub-systems. This kind of progressive proof is an alternative to classical commutation and confluence proofs.

Thus we have an extension of λ-calculus in which pattern matching is implemented via a mechanism of case analysis that behaves like a head linear substitution over constructors.We have shown that the reduction relation of $\lambda\mathcal{B}_{\mathcal{C}}$ is confluent and conservative over the λ-calculus, but also that it is complete in the sense that it provides sufficiently many reduction rules to identify all observationally equivalent normalizing terms. Surprisingly, the mechanical propagation rule "if $A//B$ and $A//C$ then $A//(B+C)$" (combined with the primitive knowledge of all commutation properties between subsystems that do not involve AppLam) is sufficient to reduce the proof of the expected 7,784 non-trivial commutation lemmas to only 12 primitive lemmas, that are established by hand. It would be interesting to investigate further to see whether the same method can be used to prove the confluence of other rewrite systems with many reduction rules -typically, systems with explicit substitutions. We started with the analysis for some of them and it seems far from being trivial. With these rewriting systems it would be interesting to define the associated (binary) closure conditions, in order to find out if they can be expressed in the same way as we have done for $\lambda\mathcal{B}_{\mathcal{C}}$. We do not know yet if the formulation of the BCC, that is, rule ... rule $\vdash$ rule is adequate for most of the known λ-calculi.

We stated and proved a topological property -separation- which is based in syntactical disagreement. The separation theorem we proved suggests that head normal forms of $\lambda\mathcal{B}_{\mathcal{C}}$ could be the adequate brick to define a notion of Böhm-tree for $\lambda\mathcal{B}_{\mathcal{C}}$ and more generally, for ML-style pattern-matching. However, the fact that it is a weak separation theorem also suggests that

the observational ordering is non-trivial on the set of normal forms. Characterizing observational ordering on normal forms could be the next step to deepen our understanding of both operational and denotational semantics of $\lambda\mathcal{B}_{\mathcal{C}}$.

Which type system for $\lambda\mathcal{B}_{\mathcal{C}}$? The reduction rules CaseApp and CaseLam which are the starting point of this work deeply challenge the traditional intuition of the notion of type, for which functions and constructed values live in different worlds. However, the good operational semantics of the calculus naturally raises the exciting question of finding a suitable type system for $\lambda\mathcal{B}_{\mathcal{C}}$.

We are also interested in improving the lemma generation technique. For instance, it could be nice to find an optimal or sub-optimal order for testing the commutation lemmas, if this is indeed plausible, with the goal of getting a minimal set of lemmas. We can also study the formulation of completeness results in relation to the inference the algorithm can perform. This is motivated by the fact that we have used just some specific lemmas which entail commutation results starting from axioms (an initial database of hand-checked results), but perhaps other rules would do as well. This would be the subject of a future research.

Last, looking for new separation theorems could be interesting. We can think of two possible lines of work. One is to ask about separation with respect to different subsystems, not the whole calculus. The other one is a different formulation of separation, namely not to use ✠ but some constant or term instead, as a device to observe and separate normal forms.

7.9 Appendix 1. Commutative Union Lemma revisited

Inspired in Lemma 7.4.1 from section 7.4, we give a new simpler proof of the well-known Hindley-Rosen's Commutative Union Lemma when considering an arbitrary family of reduction relations.

As before we denote reduction relations with capital letters such as A, A_i, etc.

Actually we will use the following equivalent formulation of Lemma 7.4.1: if $A//B_1$, $A//B_2$, $\ldots$, $A//B_k$ then $A//\cup_{1\leq i\leq k} B_i$, which easily follows by induction on k.

Lemma 7.9.1 (Hindley-Rosen's Commutative Union Lemma) *Let $\{A_i\}_{i\in I}$ be a family of reduction relations over the same set $\mathcal{A}$. If for every $i, j \in I$ $A_i//A_j$ (in particular every relation A_i is confluent), then $\cup_{i\in I}A_i$ is confluent.*

Proof: Let us denote $A = \cup_{i\in I}A_i$. Suppose $a \xrightarrow{*}_A b$ and $a \xrightarrow{*}_A c$. We will show that the diagram can be closed. Let A' be the union of the relations which were used in the above two derivations from a (thus it is a union of finite relations). Without loss of generality and to simplify notation, suppose $A' = A_1 \cup A_2 \cup \cdots \cup A_k$ for $k \geq 1$. We show below that $A'//A'$.

Take any $1 \leq i \leq k$. By hypothesis $A_i//A_j$ for every $1 \leq j \leq k$. By Lemma 7.4.1 (1), $A_i//A'$. Since i is arbitrary, use again Lemma 7.4.1 (1) and conclude that $A'//A'$. Thus the diagram is closed. □

Note that the commutation result is not applied directly to the family of relations. The reason why at the beginning of the proof we took a fixed divergence starting from an arbitrary term is that our Lemma 7.4.1, used twice in the above proof, is not intended to handle arbitrary unions of relations, just finite ones. This speaks about the fact that to close a diagram one actually can use the same relations which established it.

This proof is simpler than the classical one (e.g. in (11)) because in essence it only tiles diagrams, and it does not need to use that the reflexive-transitive closure of the union of the relations satisfies the diamond property.

Last, an analogous procedure can be used to prove the somehow weaker statement: If for every $i, j \in I$, $A_i//_w A_j$ (in particular every A_i is WCR), then $\cup_{i \in I} A_i$ is WCR.

7.10 Appendix 2. Incremental confluence proofs

Continuing our investigation on proofs of CR for subsystems using minimal axioms and conditions, we show here an alternative view of the database lemmas technique for proving commutations between rewriting subsystems given by subsets of rules.

We can describe the proofs obtained using the method developed by using proof trees associated with inference rules, in the same way we use proof trees for derivations when typing terms, as well as for proofs in many logical systems. The inference rules are the lemmas which enabled us to infer different commutation (and confluence) properties which may hold between the subsystems.

7.10.1 Confluence proof trees for CL, $\lambda_{\beta\eta}$ and $\lambda\mathcal{B}_{\mathcal{C}}$

As an experimental application of the CR proof method developed we seek shortest proofs of commuting pairs of subsystems of S, K, I-Combinatory Logic (11; 12) (in particular, $S+K+I$ and $S+K$). We measure the proof size simply as the number of tree nodes. Shortest proof trees appear on Figures 7.7 and 7.8. As a curiosity, note that this proof of confluence of $S+K+I$ does not use the confluence of $S+K$!

The method can be applied to prove confluence of the classical $\lambda_{\beta\eta}$-calculus. See Figure 7.9 for a minimal size proof tree.

We also give here proof trees for the confluence of a 4-rule subsystem of the $\lambda\mathcal{B}_{\mathcal{C}}$-calculus (the tree for the entire system should be much bigger). Thus a 15-step minimal proof tree of confluence of AppLam + AppDai + LamApp + LamDai, using 6 subsystems, is given in Figure

Let $s_1 = S$, $s_2 = K$, $s_3 = S + K$.

$$\dfrac{\dfrac{}{s_3//s_1}\,(DB) \qquad \dfrac{\dfrac{}{s_1//s_2}\,(DB) \qquad \dfrac{\dfrac{(s_2,s_2) \vdash BCC}{s_2/s_2}\,(WC) \qquad s_2 + s_2 \vdash SN}{s_2//s_2}\,(NL)}{s_3//s_2}\,(\Sigma)}{s_3//s_3}\,(\Sigma)$$

Figure 7.7: Proof tree for the confluence of $S + K$-CL

7.10, where $s_3 = \textsc{LamApp} + \textsc{LamDai}$, $s_4 = \textsc{AppDai}$, $s_8 = \textsc{AppLam}$, $s_{11} = \textsc{LamApp} + \textsc{LamDai} + \textsc{AppLam}$, $s_{12} = \textsc{AppDai} + \textsc{AppLam}$, $s_{15} = \textsc{LamApp} + \textsc{LamDai} + \textsc{AppDai} + \textsc{AppLam}$.

We seek also shortest proofs of other pairs of subsystems of this calculus, possibly using several criteria. We are interested in considering the depth of the tree as well, although we leave this and other related issues for future work.

The notation employed in these proofs is as follows. The inference rule employed in each step define the label to its right. We use (CP) each time the critical pair theorem is applied (if all critical pairs close then weak commutation holds, or WCR holds in the case both systems in the premise coincide), (DB) each time a lemma in the database is used, (NL) each time Newman's Lemma or Lemma 7.4.2 are applied, (WC) each time theorem 7.5.4 is applied in the sense that the BCC imply weak commutation (and WCR in the case both systems in the premise coincide), and (Σ) each time Lemma 7.4.1 is applied.

7.11 Some remarks

We have been exploring the existence of minimal incremental proofs of confluent systems, abstract and concrete. As a future task, we plan to investigate the problem of existence of TRSs which correspond to arbitrary closure conditions. The problem can be formulated as follows: given a finite set of ordered 3-uples of the form

$$\begin{array}{ll} r_1^1 \in s_1, r_2^1 \in s_2 & \vdash \quad r_i^1 \in s_j \\ \ldots & \\ r_1^n \in s_1, r_2^n \in s_2 & \vdash \quad r_i^n \in s_j \end{array}$$

where $i, j \in \{1, 2\}$, $i \neq j$, does there exist a TRS with rules r_i^k, $1 \leq k \leq n$, $i \in \{1, 2\}$ such that the binary closure conditions are exactly the preceding ones? We believe this is so, thus the general interest of this problem of proof synthesis is justified.

We have implemented some criterion for outputting a combined proof tree for many (sub)systems. Another criterion comes from using graphs instead of trees, thus each intermediate node ap-

Let $s_1 = S$, $s_2 = K$, $s_4 = I$, $s_5 = S + I$, $s_6 = K + I$, $s_7 = S + K + I$.

$$\dfrac{\dfrac{\dfrac{}{s_2//s_1}\,(DB) \qquad \dfrac{}{s_5//s_1}\,(DB)}{s_7//s_1}\,(\Sigma) \qquad \dfrac{\dfrac{\dfrac{}{s_1//s_2}\,(DB) \qquad \dfrac{}{s_1//s_4}\,(DB)}{s_1//s_6}\,(\Sigma) \qquad \dfrac{\dfrac{(s_6, s_6) \vdash BCC}{s_6/s_6}\,(WC) \qquad s_6 \vdash SN}{s_6//s_6}\,(NL)}{s_7//s_6}\,(\Sigma)}{s_7//s_7}\,(\Sigma)$$

Figure 7.8: Proof tree for the confluence of the full $S + K + I$-CL

$$\dfrac{\dfrac{\dfrac{}{\beta \vdash CR}\,(DB) \qquad \dfrac{}{\eta//\beta}\,(DB)}{\beta+\eta//\beta}\,(\Sigma) \qquad \dfrac{\dfrac{}{\beta//\eta}\,(DB) \qquad \dfrac{\dfrac{}{\eta \vdash WCR}\,(CP) \qquad \eta \vdash SN}{\eta \vdash CR}\,(NL)}{\beta+\eta//\eta}\,(\Sigma)}{\beta+\eta \vdash CR}\,(\Sigma)$$

Figure 7.9: Proof tree for the confluence of $\lambda_{\beta\eta}$

pears only once. We could also label some of the trees in order to avoid their repetition inside another tree.

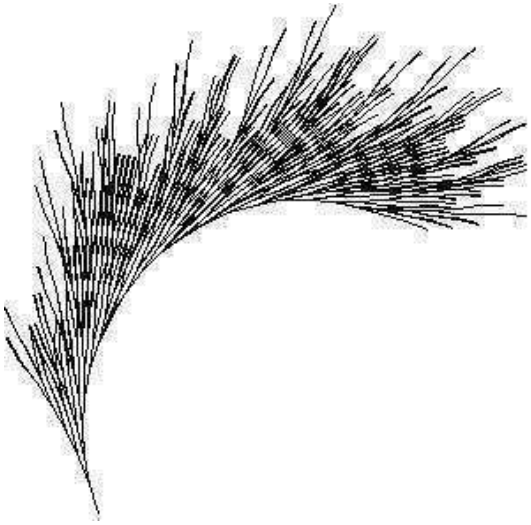
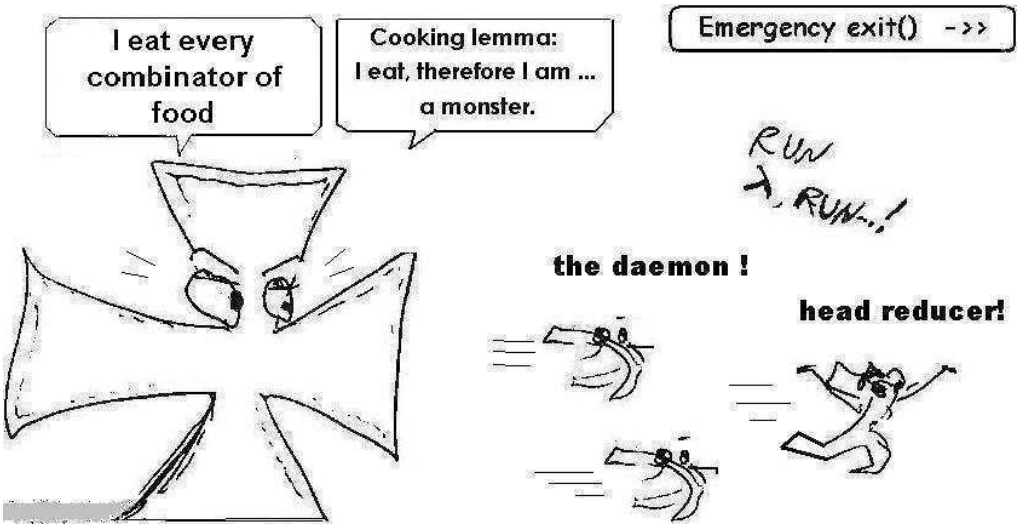


Figure 7.11: $S(S(SS)S)S(SSS)$ after 40 left-most steps



Introducing constructors to λ-calculus. More than noise.

Chapter 8

The expansion problem in lambda calculi with explicit substitution

Time to duration is as place to expansion. – J. Locke

Science may set limits to knowledge, but should not set limits to imagination. – B. Russell

ABSTRACT In the present chapter we study some specific subsystems of $\lambda\upsilon$ and λs. We analyze their respective sets of terms having the property of expanding to pure terms, as minimal sets of terms for these calculi. We prove that, contrarily to what happens in the λx-calculus in which this set of terms is trivial, for $\lambda\upsilon$ and λs it is proper and non-recursive, so it does not admit a context-free grammar and hence cannot be presented in the usual way.

8.1 Introduction

This chapter deals with specific sub-calculi of some λ-calculi concerning derivations starting from pure terms. As we pointed out on the preliminaries, a sub-calculus of a given calculus can be obtained from the original calculus starting with a subset of terms, as well as restricting the calculus to a subset of its rules, as well as restricting rule application. One can have an advantage in restricting a calculus to a specific sub-calculus, such as the fact that the set of terms may become smaller, i.e. one can discard those terms which are not needed for some specific purpose. New calculi can emerge, with possibly different properties than the original. In this chapter we address expansion problems. More precisely, the question if in a specific λ-calculus a given term would expand in 0 or more steps to some term in a given set, i.e. if there is a derivation from a member of that set to the given term. In particular, we specially consider the problem of expansion in several steps to a pure term, starting from any term from some specific λ-calculus with explicit substitution. This problem turns out to be interesting

and shows that different explicit substitution systems may have very different behavior with respect to it, as we will see, and some of these problems will result undecidable.

The motivations behind expansion to λ-calculus pure terms are several. The set of pure terms is always an important reference set when formulating simulation, projection, preservation of strong normalization, and other requirements, and in any practical implementation a substitution calculus associated to a λ-calculus with explicit substitution is required to be able to propagate substitutions when starting from pure terms -actual programs- after a β-reduction of that calculus, therefore many terms exist only to denote potential substitutions, and they will be unneeded if one restricts the attention to derivations which start from pure λ-calculus terms.

8.1.1 Related work

Expansion in explicit substitution calculi seems to have had very little treatment in the literature as far as the author knows. Polonovski (72) proves that in some calculi which enjoy the PSN property, typable terms are SN. Namely, given a term a, if there exists a pure term b such that $b \twoheadrightarrow a$, then SN of typable terms follows almost directly from PSN of the calculus and SN of simply typed λ-calculus. To achieve this, the function $Ateb(\bullet)$ is defined (its name comes from reversing $Beta$), which transforms every closure in the term into a (Beta)-redex reducing to it when this is possible (as in $\lambda\mathtt{x}$). On the other hand, when such a closure elimination by reversing (Beta) is difficult, or not possible (that will be the case when such a pure term does not exist), then some functions are defined to modify the indices in the term in such a way that the *new* term can be expanded to a pure term, but preserving typability, and then the same argument allows to achieve SN of typed terms. Nevertheless, in (72) the technique is used either when there is no pure term reducing to the term in question, or when it is difficult to find such a term. How to prove its non-existence is not discussed. Moreover, $Ateb(\bullet)$ works by inverting only the (Beta)-rule and nothing is said about inverting the other rewriting rules of the calculus.

Apart from this, expansion problems have had little or no study at all, perhaps because in some way they reflect *computing to the past* instead of computing forward in time. For instance, Waldmann (84) defines the set of ancestors of a given term in the $CL(S)$-calculus, i.e. the *combinatory logic* restricted to the use of combinator S only. Then he analyzes the rationality of the set of $CL(S)$-terms which are predecessors of the set of normal forms, and states that it is a *rational* language (i.e. it can be accepted by a finite tree automata for term algebras). Moreover, most current results about decidability of rational languages seem to be appropriate only for ground TRSs, and this is not the case of the explicit substitution calculi we study.

Our work is presented as follows. We treat the expansion problem in three calculi of explicit substitution: $\lambda\mathtt{x}$, $\lambda\upsilon$ and λs, where only the first one results a trivial problem. We show our main undecidability result for $\lambda\upsilon$, which we then transfer to λs. We finally discuss some theoretical consequences, and we conclude and suggest research directions.

8.2 Expansion in $\lambda\mathtt{x}$

The $\lambda\mathtt{x}$ calculus is a calculus where all terms expand to pure terms (in zero or more steps) as shown next.

Proposition 8.2.1 *For all $M \in \Lambda\mathtt{x}$, there exists $M' \in \Lambda$ such that $M' \underset{\lambda\mathtt{x}}{\twoheadrightarrow} M$.*

Proof: By induction on M. If $M = x$ is a variable, the result is trivial. If $M = PQ$, then by inductive hypothesis (IH) there exists $P' \in \Lambda$ such that $P' \underset{\lambda\mathtt{x}}{\twoheadrightarrow} P$, and there exists $Q' \in \Lambda$ such that $Q' \underset{\lambda\mathtt{x}}{\twoheadrightarrow} Q$. Take $M' = P'Q' \underset{\lambda\mathtt{x}}{\twoheadrightarrow} M$. If $M = \lambda x.P$, then by IH there exists $P' \in \Lambda$ such that $P' \underset{\lambda\mathtt{x}}{\twoheadrightarrow} P$. Take $M' = \lambda x.P' \underset{\lambda\mathtt{x}}{\twoheadrightarrow} M$. If $M = P\langle x := Q\rangle$, then by IH there exists $P' \in \Lambda$ such that $P' \underset{\lambda\mathtt{x}}{\twoheadrightarrow} P$, and there exists $Q' \in \Lambda$ such that $Q' \underset{\lambda\mathtt{x}}{\twoheadrightarrow} Q$. Take $M' = (\lambda x.P')Q' \underset{\lambda\mathtt{x}}{\twoheadrightarrow} M$. □

Remark 8.2.2 *Note that the (Beta)-rule becomes necessary in the above result when the term M is not pure. Moreover, no other rule is used. The same holds for the $\lambda\mathtt{x}^{-}$ calculus.*

8.3 Expansion in $\lambda\upsilon$

In order to treat the minimality of the $\lambda\upsilon$-terms set and related problems, a study of the expansion should be done. Expansion permits to know the "past" of a term. In particular we are interested in terms with a "pure" past, i.e. those expanding to a classical de Bruijn term. The purpose of this section is to show that not all terms have this property, and to capture them, as well as to present a suitable calculus having the same good properties of $\lambda\upsilon$ where all terms expand to pure terms.

We will see that the above mentioned set of terms is a proper subset of Λ^t_υ, by the exhibition of families of examples of $\lambda\upsilon$-terms which are not in this set. Next we will give the description we found for these terms, and we will treat the problem of deciding whether such an expansion exists.

Remark that the (Beta)-rule becomes necessary to recover pure terms from closures when doing expansion. This is because all left-hand sides of the υ-rules include some closure thus they are not pure terms. The same holds in $\lambda\mathtt{x}$, λs and most explicit substitution calculi. This motivates the inclusion of the (Beta)-rule in the study of expansion from now onwards, that is, to consider not υ but $\lambda\upsilon$.

Due to these considerations, we find interesting to treat expansion in the full calculus.

Definition 8.3.1 ($\lambda\upsilon$-terms with pure expansion) *Given $M \in \Lambda_\upsilon^t$ we say that M has a pure expansion if there exists $N \in \Lambda$ such that $N \underset{\lambda\upsilon}{\twoheadrightarrow} M$. Let $\Lambda_\upsilon^p = \mathcal{S}(\Lambda) = \{M \in \Lambda_\upsilon^t \mid M$ has a pure expansion$\}$.*

8.3.1 Terms with pure expansion

The next sections of this chapter can be read independently of the following part of this section. So it is possible to skip the subsequent definitions and results in this section and go directly to section 8.4.

Next we aim to prove that in $\lambda\upsilon$ there are (big families of) terms which do not expand to pure terms. For this we introduce a notion of "good context" in this calculus. From now onwards unless explicitly stated, all contexts will be term contexts.

Definition 8.3.2 *A context $C = C\{\Box\}$ is of λ-type if $C\{\Box\} = D\{\lambda D'\{\Box\}\}$ with D, D' contexts.*

Definition 8.3.3 *A context $C = C\{\Box\}$ is of /-type if $C\{\Box\} = D\{(D'\{\Box\})[Q/]\}$ with D, D' contexts and $Q \in \Lambda_\upsilon^t$.*

Definition 8.3.4 *A context will be called* good *if it is not of λ-type nor of /-type.*

In other terms, the hole is not under the scope of a λ nor inside the *head* of a closure of /-type.

Remark that a sub-context of a good context is also good.

Definition 8.3.5 *For $a \in \Lambda_\upsilon^t$ we define the number of lifts in a, written $n_\Uparrow(a)$, in the expected way:*

$$\begin{array}{lcllcl} n_\Uparrow(n) & = & 0 & n_\Uparrow(a/) & = & n_\Uparrow(a) \\ n_\Uparrow(ab) & = & n_\Uparrow(a) + n_\Uparrow(b) & n_\Uparrow(\lambda a) & = & n_\Uparrow(a) \\ n_\Uparrow(a[s]) & = & n_\Uparrow(a) + n_\Uparrow(s) & n_\Uparrow(\uparrow) & = & 0 \\ n_\Uparrow(\Uparrow(s)) & = & n_\Uparrow(s) + 1 & & & \end{array}$$

or equivalently

$$\begin{array}{lcllcl} n_\Uparrow(n) & = & 0 & & & \\ n_\Uparrow(ab) & = & n_\Uparrow(a) + n_\Uparrow(b) & n_\Uparrow(\lambda a) & = & n_\Uparrow(a) \\ n_\Uparrow(a[\Uparrow^k(b/)]) & = & n_\Uparrow(a) + n_\Uparrow(b) + k & n_\Uparrow(a[\Uparrow^k(\uparrow)]) & = & n_\Uparrow(a) + k \end{array}$$

For $a \in \Lambda_\upsilon^t$ we define the number of shifts in a, written $n_\uparrow(a)$, in the expected way:

$$\begin{array}{lcllcl} n_\uparrow(n) & = & 0 & n_\uparrow(a/) & = & n_\uparrow(a) \\ n_\uparrow(ab) & = & n_\uparrow(a) + n_\uparrow(b) & n_\uparrow(\lambda a) & = & n_\uparrow(a) \\ n_\uparrow(a[s]) & = & n_\uparrow(a) + n_\uparrow(s) & n_\uparrow(\uparrow) & = & 1 \\ n_\uparrow(\Uparrow(s)) & = & n_\uparrow(s) & & & \end{array}$$

or equivalently

$$\begin{array}{llllll} n_\uparrow(n) & = & 0 & & & \\ n_\uparrow(ab) & = & n_\uparrow(a)+n_\uparrow(b) & n_\uparrow(\lambda a) & = & n_\uparrow(a) \\ n_\uparrow(a[\Uparrow^k (b/)]) & = & n_\uparrow(a)+n_\uparrow(b) & n_\uparrow(a[\Uparrow^k (\uparrow)]) & = & n_\uparrow(a)+1 \end{array}$$

The idea behind good contexts is that they are "potentially impure", i.e. when they expand they somehow cannot reduce the number of $\Uparrow$ in a term.

In the following proofs, for any term M, when a context C is "clear from the context", by abuse of notation we will freely write statements like "$\Box \in M$" which will mean that within the context C the position of the hole is located below the position of the sub-term M. For example, if for some term e, $C\{e\} = a[s]$, when we speak about the position of C's hole, unless $C\{\Box\} = \Box$, we will consider it "to be in a" (resp. "in s"), and we will write $\Box \in a$ (resp. $\Box \in s$), if that position has the form $1.q$ (resp. $2.q$).

The reason of identifying good contexts is the following

Lemma 8.3.6 (invariance of $\Uparrow -/$ in good contexts) *Let C be a good context, $B, M, N \in \Lambda^t_\upsilon, k \geq 1$, such that $B \underset{\lambda\upsilon}{\longrightarrow} C\{M[\Uparrow^k (N/)]\}$. Then there exists a good context C', there exist $M', N' \in \Lambda^t_\upsilon$, and $k' \geq 1$ such that $B = C'\{M'[\Uparrow^{k'} (N'/)]\}$ (actually $k' = k$ or $k' = k+1$).*

PROOF: By induction on the context C. Let us call $e = M[\Uparrow^k (N/)]$. We will not give M', N', k' when the choice is clear once C' has been chosen.

- If $C\{\Box\} = \Box$ with $B \underset{\lambda\upsilon}{\longrightarrow} e$.
  - if the reduction is at the root of B:
    - if the reduction is $B = (\lambda a)b \to a[b/] = e$ a (Beta)-step, it would imply $k = 0$ but this is not possible by the hypothesis $k \geq 1$.
    - $B = (ab)[s] \to a[s]b[s] = e$ cannot happen since e does not match an application.
    - $B = (\lambda a)[s] \to \lambda(a[\Uparrow (s)]) = e$ cannot happen since e does not match an abstraction.
    - $B = 1[\Uparrow (s)] \to 1 = e$ cannot happen since e does not match the reduct.
    - $B = (n+1)[\Uparrow (s)] \to n[s][\uparrow] = e$ cannot happen since e does not match the reduct.
    - $B = n[\uparrow] \to n+1 = e$ cannot happen since e does not match the reduct.
    - $B = 1[a/] \to a = e$, take $C'\{\Box\} = 1[\Box/]$ which is a good context, $M' = M$ and $k' = k$.
    - $B = (n+1)[a/] \to n = e$ cannot happen since e does not match the reduct.
  - if the reduction is internal in B:
    - $a'b \to ab = e$ with $a' \to a$, it cannot happen since e does not match the reduct.

* $ab' \to ab = e$ with $b' \to b$, it cannot happen since e does not match the reduct.
* $\lambda a' \to \lambda a = e$ with $a' \to a$, it cannot happen since e does not match the reduct.
* $M'[\Uparrow^k (N/)] \to M[\Uparrow^k (N/)] = e$ with $M' \to M$, immediate taking $C'\{\Box\} = \Box$ which is a good context.
* $M[\Uparrow^k (N'/)] \to M[\Uparrow^k (N/)] = e$ with $N' \to N$, immediate taking $C'\{\Box\} = \Box$ which is a good context.

- If $C \neq \Box$:

1. if the reduction is at the root of B:
   (a) $B = (\lambda a)b \to a[b/] = C\{e\}$.
   $\Box \in a$, it cannot happen since C is a good context
   $\Box \in b$ then take $C'\{\Box\} = (\lambda a)D\{\Box\}$, where $b = D\{e\}$ with D good, thus C' is good.
   (b) $B = (ab)[s] \to a[s]b[s]$, we have the following cases:
   – if $\Box \in a$, take $C'\{\Box\} = (D\{\Box\}b)[s]$ where $a = D\{e\}$ with D good, thus C' is good.
   – if $\Box \in b$, take $C'\{\Box\} = (aD\{\Box\})[s]$ where $b = D\{e\}$ with D good, thus C' is good.
   – if $\Box \in s$ at the left (i.e. its position is of the form $1.2.q$ for some q), then $s = \Uparrow^k (P/)$ for some $k \geq 0$ and P, take $C'\{\Box\} = a[\Uparrow^k (D\{\Box\}/)]b[s]$ where $P = D\{e\}$ with D good, thus C' is good.
   – if $\Box \in s$ at the right (i.e. its position is of the form $2.2.q$ for some q), then again $s = \Uparrow^k (P/)$ for some $k \geq 0$ and P, take $C'\{\Box\} = a[s]b[\Uparrow^k (D\{\Box\}/)]$ where $P = D\{e\}$ with D good, thus C' is good.
   – if $\Box$ occurs at the root of $a[s] = e$, take $C'\{\Box\} = \Box b[s]$ which is good.
   – if $\Box$ occurs at the root of $b[s] = e$, take $C'\{\Box\} = a[s]\Box$ which is good.
   (c) $B = (\lambda a)[s] \to \lambda(a[\Uparrow (s)]) = C\{e\}$, it cannot happen by hypothesis because C must not begin with λ.
   (d) $B = 1[a/] \to a = C\{e\}$, take $C'\{\Box\} = 1[C\{\Box\}/]$ which is good.
   (e) $B = (n+1)[a/] \to n$, it cannot happen, since $n \neq C\{e\}$ for any context C.
   (f) $B = 1[\Uparrow (s)] \to 1$, it cannot happen, since $1 \neq C\{e\}$ for any context C.
   (g) $B = (n+1)[\Uparrow (s)] \to n[s][\uparrow] = C\{e\}$, the hole cannot occur at the root since e does not match $n[s][\uparrow]$, then
   – if $\Box \in s$, take $C'\{\Box\} = (n+1)[\Uparrow (D\{\Box\})]$, where $s = D\{e\}/$, thus C' is good.

- if $n[s]$ occurs at the position of $\Box$, i.e. $n[s] = M[\Uparrow^k (N/)]$, then $s = \Uparrow^k (N/)$, then $B = (n+1)[\Uparrow^{k+1} (N/)]$, take $k' = k+1$, $M' = n+1$, $N' = N$ and $C' = \Box$ which is good.

(h) $B = n[\uparrow] \to n+1$, it cannot happen, since $n+1 \neq C\{e\}$.

2. if the reduction is internal in B:

(a) if $B = a'b \to ab = C\{e\}$ with $a' \to a$ we have:

- if $\Box \in b$, straightforward: take $C'\{\Box\} = a'D\{\Box\}$ where $b = D\{e\}$ with D good, thus C' is good
- if $\Box \in a$, by IH $a' = D'\{M'[\Uparrow^{k'} (N'/)]\}$ with D' good, take $C'\{\Box\} = D'\{\Box\}b$ which is good.

(b) if $B = ab' \to ab$ with $b' \to b$, it is analogous to the previous case. We have:

- if $\Box \in b$, by IH $b' = D'\{M'[\Uparrow^{k'} (N'/)]\}$ with D' good, take $C'\{\Box\} = aD'\{\Box\}$ which is good.
- if $\Box \in a$, straightforward: take $C'\{\Box\} = D\{\Box\}b'$ where $a = D\{e\}$ with D good, thus C' is good.

(c) if $B = \lambda a' \to \lambda a = C\{e\}$ thus $\Box \in a$, but C' is good so this cannot happen.

(d) if $B = a'[\Uparrow^r (b/)] \to a[\Uparrow^r (b/)]$ with $a' \to a$ and $r > 0$, then

- if $\Box \in a$, use IH and take $C' = D'\{\Box\}[\Uparrow^r (b/)]$ where $a' = D'\{M'[\Uparrow^{k'} (N'/)]\}$ with D' good, thus C' is good.
- if $\Box \in b$, take $C' = a'[\Uparrow^r (D\{\Box\}/)]$ where $b = D\{e\}$ with D good, thus C' is good.
- $\Box$ cannot be at the position of $\Uparrow^m (b/)$ for some $0 \leq m \leq r$ since C is a term context.

(e) if $B = a[\Uparrow^r (b'/)] \to a[\Uparrow^r (b/)]$ with $b' \to b$ and $r > 0$, then

- if $\Box \in a$, take $C'\{\Box\} = D\{\Box\}[\Uparrow^r (b'/)]$ where $a = D\{e\}$ with D good, thus C' is good.
- if $\Box \in b$, use IH and take $C' = a[\Uparrow^r (D'\{\Box\}/)]$ where $b' = D'\{M'[\Uparrow^{k'} (N'/)]\}$ with D' good, thus C' is good.
- as before $\Box$ cannot be at the position of $\Uparrow^m (b/)$ for some $0 \leq m \leq r$ since C is a term context.

(f) if $B = a'[b/] \to a[b/]$ with $a' \to a$ (i.e. $B \to a[\Uparrow^r (b/)]$ with $r = 0$) we have:

- if $\Box \in b$, take $C'\{\Box\} = a'[D\{\Box\}/]$ where $b = D\{e\}$ with D good, thus C' is good
- if $\Box \in a$, it cannot happen because C is good.

- remember that $\Box$ cannot be at the root of $a[b/]$ because we are considering the case $C \neq \Box$.

(g) if $B = a[b'/] \to a[b/]$ with $b' \to b$ we have:
- if $\Box \in a$, it cannot happen because C is good.
- if $\Box \in b$, then use IH and take $C'\{\Box\} = a[D'\{\Box\}/]$ where $b' = D'\{M'[\Uparrow^{k'} (N'/)]\}$ with D' good, thus C' is good.
- again, $\Box$ cannot be at the root of $a[b/]$ because $C \neq \Box$.

(h) if $B = a'[\Uparrow^r (\uparrow)] \to a[\Uparrow^r (\uparrow)]$ with $a' \to a$ and $r \geq 0$, then $\Box$ cannot occur at the root, so we have that:
- $\Box \in a$, use IH and take $C'\{\Box\} = D'\{\Box\}[\Uparrow^r (\uparrow)]$ where $a' = D'\{M'[\Uparrow^{k'} (N'/)]\}$ with D' good, then C' is good.

$\Box$

Lemma 8.3.7 (invariance of $\Uparrow - \uparrow$ in good contexts) *Let C be a good context, $B, M \in \Lambda^t_\upsilon, k \geq 0$, such that $B \underset{\lambda\upsilon}{\longrightarrow} C\{M[\Uparrow^k (\uparrow)]\}$. Then there exist a good context C', $M' \in \Lambda^t_\upsilon$, $s' \in \Lambda^s_\upsilon$ and $k' \geq 0$ such that $B = C'\{M'[\Uparrow^{k'} (s')]\}$, with:*

1. *either $k' \geq 0$ and $s' = \uparrow$*
2. *or $k' \geq 1$ and $s' = N/$ for some $N \in \Lambda^t_\upsilon$.*

Proof: By induction on the context C. Let us call $e = M[\Uparrow^k (\uparrow)]$. As before, we will not give M', s', k' when the choice is clear once C' has been chosen.

- If $C\{\Box\} = \Box$ then $B \underset{\lambda\upsilon}{\longrightarrow} e$.
  - if the reduction is at the root of B:
    - if the reduction is $B = (\lambda a)b \to a[b/] = e$ a (Beta)-step, it would imply $\Uparrow^k (\uparrow) = b/$ which is not possible.
    - $B = (ab)[s] \to a[s]b[s] = e$ cannot be since e does not match an application.
    - $B = (\lambda a)[s] \to \lambda(a[\Uparrow (s)]) = e$ cannot be since e does not match an abstraction.
    - $B = 1[\Uparrow (s)] \to 1$ cannot happen since e does not match the reduct.
    - $B = (n+1)[\Uparrow (s)] \to n[s][\uparrow]$ then the left term has the desired form for any substitution s, taking $C'\{\Box\} = \Box$ which is good, where $k' \geq 1$.
    - $B = n[\uparrow] \to n + 1 = e$, it cannot happen since e does not match the reduct.
    - $B = 1[a/] \to a = e$, take $C'\{\Box\} = 1[\Box/]$ which is good.
    - $B = (n+1)[a/] \to n = e$, it cannot happen since e does not match the reduct.
  - if the reduction is internal in B, the only possibility is

* $M'[\Uparrow^k(\uparrow)] \to M[\Uparrow^k(\uparrow)]$ with $M' \to M$, thus take $C' = \Box$ which is good.

- If $C\{\Box\} \neq \Box$:

1. if the reduction is at the root of B:
   (a) $B = (\lambda a)b \to a[b/] = C\{e\}$.
   $\Box \in a$, it cannot happen since C is good
   $\Box \in b$ then take $C'\{\Box\} = (\lambda a)D\{\Box\}$, where $b = D\{e\}$, thus C' is good.
   (b) $B = (ab)[s] \to a[s]b[s]$, we have the following cases:
   – if $\Box \in a$, take $C'\{\Box\} = (D\{\Box\}b)[s]$ where $a = D\{e\}$, thus C' is good.
   – if $\Box \in b$, take $C'\{\Box\} = (aD\{\Box\})[s]$ where $b = D\{e\}$, thus C' is good.
   – if $\Box \in s$ at the left (i.e. its position is of the form $1.2.q$ for some q), then $s = \Uparrow^k (P/)$ for some $k \geq 0$ and P, take $C'\{\Box\} = a[\Uparrow^k (D\{\Box\}/)]b[s]$ where $P = D\{e\}$, thus C' is good because D is good.
   – if $\Box \in s$ at the right (i.e. its position is of the form $2.2.q$ for some q), then again $s = \Uparrow^k (P/)$ for some $k \geq 0$ and P, take $C'\{\Box\} = a[s]b[\Uparrow^k (D\{\Box\}/)]$ where $P = D\{e\}$, thus C' is good because D is good.
   – if $\Box$ occurs at the root of $a[s] = e$, take $C'\{\Box\} = \Box b[s]$ which is good.
   – if $\Box$ occurs at the root of $b[s] = e$, take $C'\{\Box\} = a[s]\Box$ which is good.
   (c) $B = (\lambda a)[s] \to \lambda(a[\Uparrow (s)]) = C\{e\}$, it cannot happen by hypothesis because C is good.
   (d) $B = 1[a/] \to a = C\{e\}$, take $C'\{\Box\} = 1[C\{\Box\}/]$ which is good.
   (e) $B = (n+1)[a/] \to n$, it cannot happen, since $n \neq C\{e\}$ for any context C.
   (f) $B = 1[\Uparrow (s)] \to 1$, it cannot happen, since $1 \neq C\{e\}$ for any context C.
   (g) $B = (n+1)[\Uparrow (s)] \to n[s][\uparrow] = C\{e\}$, the hole cannot occur at the root so there are three cases:
   – if $\Box \in s$, take $C'\{\Box\} = (n+1)[\Uparrow^{k+1} (D\{\Box\}/)]$ where $s = \Uparrow^k (D\{e\}/)$.
   – if n occurs at the position of $\Box$, i.e. $n = e = M[\Uparrow^k (\uparrow)]$, which is not possible.
   – if $n[s]$ occurs at the position of $\Box$, i.e. $n[s] = e = M[\Uparrow^k (\uparrow)]$, then $s = \Uparrow^k (\uparrow)$ thus $B = (n+1)[\Uparrow^{k+1} (\uparrow)]$, take $k' = k+1$, $M' = n+1$ and $C' = \Box$ which is good.
   (h) $B = n[\uparrow] \to n+1$, it cannot happen, since $n+1 \neq C\{e\}$.
2. if the reduction is internal in B:
   (a) if $B = a'b \to ab = C\{e\}$ with $a' \to a$ we have:
   – if $\Box \in b$, straightforward: take $C'\{\Box\} = a'D\{\Box\}$ where $b = D\{e\}$ with D good, thus C' is good.

- if $\Box \in a$, by IH $a' = D'\{M'[\Uparrow^{k'} (s')]\}$ with D' good, take $C'\{\Box\} = D'\{\Box\}b$ which is good and the other conditions hold.

(b) if $B = ab' \to ab$ with $b' \to b$, it is analogous to the previous case. We have:
- if $\Box \in b$, by IH $b' = D'\{M'[\Uparrow^{k'} (s')]\}$ with D' good, take $C'\{\Box\} = aD'\{\Box\}$ which is good and the other conditions hold.
- if $\Box \in a$, straightforward: take $C'\{\Box\} = D\{\Box\}b'$ where $a = D\{e\}$ with D good, thus C' is good.

(c) if $B = \lambda a' \to \lambda a = C\{e\}$ thus $\Box \in a$, but C' is good so this cannot be the case.

(d) if $B = a'[\Uparrow^r (b/)] \to a[\Uparrow^r (b/)]$ with $a' \to a$ and $r > 0$, then
- if $\Box \in a$, use IH and take $C'\{\Box\} = D'\{\Box\}[\Uparrow^r (b/)]$ where $a' = D'\{M'[\Uparrow^{k'} (s')]\}$ with D' good, thus C' is good and the other conditions hold.
- if $\Box \in b$, take $C'\{\Box\} = a[\Uparrow^r (D\{\Box\}/)]$ where $b = D\{e\}$ with D good, thus C' is good.

(e) if $B = a[\Uparrow^r (b'/)] \to a[\Uparrow^r (b/)]$ with $b' \to b$ and $r > 0$, then
- if $\Box \in a$, take $C'\{\Box\} = D\{\Box\}[\Uparrow^r (b'/)]$ where $a = D\{e\}$ with D good, thus C' is good.
- if $\Box \in b$, use IH and take $C'\{\Box\} = a[\Uparrow^r (D'\{\Box\}/)]$ where $b' = D'\{M'[\Uparrow^{k'} (s')]\}$ with D' good, thus C' is good and the other conditions hold.

(f) if $B = a'[b/] \to a[b/]$ with $a' \to a$ (i.e. $B \to a[\Uparrow^r (b/)]$ with $r = 0$) we have:
- if $\Box \in b$, take $C'\{\Box\} = a'[D\{\Box\}/]$ where $b = D\{e\}$ with D good, thus C' is good
- if $\Box \in a$, it cannot happen because C is good.
- $\Box$ cannot be at the root of $a[b/]$ because we are considering the case $C \neq \Box$.

(g) if $B = a[b'/] \to a[b/]$ with $b' \to b$ we have:
- if $\Box \in a$, it cannot happen because C is good.
- if $\Box \in b$, then use IH and take $C'\{\Box\} = a[D'\{\Box\}/]$ where $b' = D'\{M'[\Uparrow^{k'} (s')]\}$ with D' good, thus C' is good and the other conditions hold.
- again, $\Box$ cannot be at the root of $a[b/]$ because $C \neq \Box$.

(h) if $B = a'[\Uparrow^r (\uparrow)] \to a[\Uparrow^r (\uparrow)]$ with $a' \to a$ and $r \geq 0$, then the hole cannot occur at the root, so we have that:
- $\Box \in a$, use IH and take $C'\{\Box\} = D'\{\Box\}[\Uparrow^r (\uparrow)]$ where $a' = D'\{M'[\Uparrow^{k'} (s')]\}$ with D' good, then C' is good and the other conditions hold.

□

As a consequence of the previous two lemmas we have the following

Proposition 8.3.8 *1. If $B \longrightarrow\!\!\!\!\!\rightarrow C\{M[\Uparrow^k (N/)]\}$ with C a good context and $k \geq 1$, then there exists C' a good context, there exist $M', N', k' \geq 1$ such that $B = C'\{M'[\Uparrow^{k'} (N'/)]\}$.*

2. If $B \longrightarrow\!\!\!\!\!\rightarrow C\{M[\Uparrow^k (\uparrow)]\}$ with C a good context and $k \geq 0$, then there exists C' a good context, there exist $M', N', k' \geq 0$ and $s' \in \Lambda_\upsilon^s$ such that $B = C'\{M'[\Uparrow^{k'} (s')]\}$ and

(a) either $k' \geq 0$ and $s' = \uparrow$

(b) or $k' \geq 1$ and $s' = N/$ for some $N \in \Lambda_\upsilon^t$.

Proof: Both items are proved by induction on the length of the derivation, using lemmas 8.3.6 and 8.3.7. □

Let us remark that if one states both items at once, e.g., if $B \longrightarrow\!\!\!\!\!\rightarrow C\{M[\Uparrow^k (s)]\}$ with C a good context and $k \geq 0$ then there exists C' a good context, $M', k' \geq 0$ and $s' \in \Lambda_\upsilon^s$ such that $B = C'\{M'[\Uparrow^{k'} (s')]\}$, then one loses the property since the last closure would not have a uniform structure. The (FVarLift)-rule is the only reason for the double condition at the conclusion of Lemma 8.3.7.

In particular we have the following Corollary, yielding two families of infinite terms not expanding to pure terms.

Corollary 8.3.9 *In $\lambda\upsilon$, the terms of the form $M[\Uparrow^k (N/)]$ with $k \geq 1$ and the terms of the form $M[\Uparrow^k (\uparrow)]$ with $k \geq 0$ do not expand to pure terms.*

Proof: By Proposition 8.3.8, any expansion of such a term will have the form $B = C'\{M'[\Uparrow^{k'} (s')]\}$, then $n_{\Uparrow}(B) \geq k' \geq 1$ or $n_{\uparrow}(B) \geq 1$, in either case $B \notin \Lambda$. □

These families of terms are not exhaustive, as we will see in the following section. Now we can state the following

Corollary 8.3.10 *$(\Lambda_\upsilon^p, \rightarrow_{\lambda\upsilon}|_{\Lambda_\upsilon^p})$, where $\rightarrow_{\lambda\upsilon}|_{\Lambda_\upsilon^p}$ is the restriction of $\rightarrow_{\lambda\upsilon}$ to Λ_υ^p, is a proper sub-ARS of $\lambda\upsilon$, with set of terms $\Lambda_\upsilon^p = \mathcal{S}(\Lambda)$.*

Proof: We have proved that the set of terms is proper. We now show that, if $M \in \Lambda_\upsilon^p$ and $M \rightarrow_{\lambda\upsilon} N$, then $N \in \Lambda_\upsilon^p$. Since $M \in \Lambda_\upsilon^p$, there exists M' in Λ such that $M' \underset{\lambda\upsilon}{\twoheadrightarrow} M$. Then $M' \underset{\lambda\upsilon}{\twoheadrightarrow} N$, so N expands to a (the same) pure term, thus $N \in \Lambda_\upsilon^p$. It is immediate that $\Lambda_\upsilon^p = \mathcal{S}(\Lambda)$. □

Let us call $\lambda\upsilon^p$ the newly defined sub-calculus of $\lambda\upsilon$.

One consequence of the previous analysis is that $\lambda\upsilon^p$ has a set of terms which is strictly included in the set Λ_υ^t. Something analogous will happen with other calculi, and we leave this for section 8.5 (as well as for future work).

8.3.2 Some applications on mappings between calculi

In this last part of the section we show an application of Corollary 8.3.9 dealing with the existence of appropriate mappings between the calculi. For the sole purpose of this subsection a good mapping will be a function with minimal preservation properties as given by the following definition. Note in this subsection we will not distinguish classical λ-terms with names and λ-terms with de Bruijn indices, so for instance when we write the set Λ we will mean the classical set of λ-terms or the set of de Bruijn terms, which will be given by the context.

Definition 8.3.11 (good mapping) *Suppose we have a mapping from the term set of a calculus $\lambda\zeta_1$ to the term set of another calculus $\lambda\zeta_2$, given by a function $t : \Lambda\zeta_1 \to \Lambda\zeta_2$, where $\Lambda\zeta_1$ and $\Lambda\zeta_2$ are their respective sets of terms. Let us assume that $\Lambda \subset \Lambda\zeta_i$ for $i = 1, 2$, that is, these calculi include as terms the pure λ-terms (either classical or de Bruijn) as usual. We will call t a* good mapping *(or a homomorphism) from $\lambda\zeta_1$ to $\lambda\zeta_2$ iff the following two (expected) conditions are met:*

1. *$t_{|\Lambda} = id_\Lambda$, i.e. t on any pure term yields the same term*
2. *if $M \to_{\lambda\zeta_1} N$ then $t(M) \underset{\lambda\zeta_2}{\twoheadrightarrow} t(N)$, i.e. t preserves reduction in a weak sense*

Remark that condition (2) in Definition 8.3.11 is very weak (eg. weaker than the usual Simulation) because we do not distinguish between rewriting rules of any calculus and the only requirement is that every rewriting step of one calculus is just mapped into (0 or more) rewriting steps of any nature in the second calculus. A trivial example for condition (2) is a constant function. But such a function is not a good mapping since it does not satisfy condition (1) – all pure terms must be mapped to themselves (or to their de Bruijn translations) and not to a unique term. Thus both conditions seem natural. Condition (1) is often valid on mappings used in the literature.

Furthermore, the requirement for a good mapping to be surjective, or injective, can be reasonable in the context of explicit substitution calculi, for the following reason. Without this requirement, in the presence of calculi satisfying SN of the associated substitution calculus, Projection and Simulation, the function given by taking the substitution calculus normal form is always a good mapping. More precisely:

Example 8.3.12 *Recall the function $x : \Lambda\mathtt{x} \to \Lambda$ defined as the x-normal form of terms, but considered as a function $x : \Lambda\mathtt{x} \to \Lambda^t_\upsilon$. This application is a good mapping from $\lambda\mathtt{x}$ to $\lambda\upsilon$.*

Proof: Condition (1) clearly holds since $x(M) = M$ for every $M \in \Lambda$. To verify condition (2), let $M \to_{\lambda\mathtt{x}} N$. Then by Projection of $\lambda\mathtt{x}$, $x(M) \underset{\beta}{\twoheadrightarrow} x(N)$. Since both are pure terms, and $\lambda\upsilon$ satisfies Simulation of the β-reduction, $x(M) \underset{\lambda\upsilon}{\twoheadrightarrow} x(N)$. □

Thus it seems appropriate to require good mappings to be onto. As an example of a surjective good mapping from λs to $\lambda\upsilon$ take the function T which appears in section 8.6. But clearly the mapping $x : \Lambda\mathtt{x} \to \Lambda_\upsilon^t$ of Example 8.3.12 is not surjective nor injective. We pose the question whether, given a pair of calculi, surjective and/or injective good mappings exist from one of them to the other one. As a partial answer we give this

Proposition 8.3.13 *There are no surjective good mappings from $\lambda\mathtt{x}$ to $\lambda\upsilon$.*

Proof: Suppose $t : \Lambda\mathtt{x} \to \Lambda_\upsilon^t$ is a surjective good mapping. Take any $a \in \Lambda_\upsilon^t$. Since t is surjective, there exists $M \in \Lambda\mathtt{x}$ such that $t(M) = a$. By Proposition 8.2.1 there exists $P \in \Lambda$ such that $P \underset{\lambda\mathtt{x}}{\twoheadrightarrow} M$. By condition (2), $t(P) \underset{\lambda\upsilon}{\twoheadrightarrow} t(M)$, but by condition (1) $t(P) = P$. Since a was an arbitrary $\lambda\upsilon$-term, we have that for every $a \in \Lambda_\upsilon^t$ there exists $P \in \Lambda_{dB}$ such that $P \underset{\lambda\upsilon}{\twoheadrightarrow} a$, which contradicts Corollary 8.3.9. □

8.4 Undecidability results for $\lambda\upsilon$

We now prepare for the proof of our main result, which states that the set Λ_υ^p is non-recursive. We recall Λ_{dB} the set of de Bruijn terms and the β_{dB}-reduction in the usual way. In what follows we just use Λ and β for short, when their meanings are clear from the context.

We begin with the following well-known theorem of D. Scott allowing to identify some non-recursive (i.e. undecidable) sets of terms in λ-calculus.

Proposition 8.4.1 (Scott's theorem) *Let $C \subseteq \Lambda$ a proper subset of λ-terms (i.e. not empty and not all Λ). If C is closed under the $=_\beta$ relation (i.e., if $M \in C$, $N \in \Lambda$ and $M =_\beta N$ then $N \in C$), then C is non-recursive.*

Proof: See (12) for a proof for classical λ-calculus. For the λ-calculus in the de Bruijn setting, the same holds due to the isomorphism (see the preliminaries) which is a computable function. □

Remark that Scott's theorem is analogous to Rice Theorem for sets of computable functions. As an important consequence, we have:

Lemma 8.4.2 *Let $N \in \Lambda$ be a fixed term. Let $C = \{M \in \Lambda \mid M =_\beta N\}$ i.e. the set of all terms β-equivalent to N. Then C is non-recursive.*

Proof: Using Scott's theorem since this set is clearly non-trivial and closed under $=_\beta$. □

Although it is well-known, we state the following

Corollary 8.4.3 *The $=_\beta$ relation is undecidable.*

PROOF: Suppose it is decidable, thus there exists an algorithm for testing $M =_\beta N$ for every pair of terms $M, N \in \Lambda$. Then the set $\{M \in \Lambda \mid M =_\beta 1\}$ would be recursive since we could use the algorithm for the pair $M, 1$, which is absurd by the previous Lemma taking $N = 1$. □

Also we have:

Corollary 8.4.4 *The* $\underset{\beta}{\twoheadrightarrow}$ *relation is undecidable.*

PROOF: Suppose it is decidable, thus there exists an algorithm for testing $M \underset{\beta}{\twoheadrightarrow} N$ for every pair of terms $M, N \in \Lambda$. Take the set $C = \{M \in \Lambda \mid M \underset{\beta}{\twoheadrightarrow} 1\}$. Then C would be recursive since we could use the algorithm for the pair $M, 1$. But clearly $C = \{M \in \Lambda \mid M =_\beta 1\}$ since 1 is a β-normal form. This is absurd by Lemma 8.4.2. □

Since our aim is to study expansion, we add the following consequence (not mentioned at least in the classical literature of λ-calculus):

Corollary 8.4.5 *In* λ*-calculus the* common expansion *problem is undecidable. I.e., given* $M, N \in \Lambda$*, the problem of deciding if there exists* $P \in \Lambda$ *such that* $P \underset{\beta}{\twoheadrightarrow} M$ *and* $P \underset{\beta}{\twoheadrightarrow} N$ *is undecidable.*

PROOF: Suppose it is decidable. Take the set $C = \{M \in \Lambda \mid \exists P \in \Lambda\ P \underset{\beta}{\twoheadrightarrow} M,\ P \underset{\beta}{\twoheadrightarrow} 1\}$. The statement $\exists P \in \Lambda\ P \underset{\beta}{\twoheadrightarrow} M,\ P \underset{\beta}{\twoheadrightarrow} 1$ is equivalent to the statement $M \underset{\beta}{\twoheadrightarrow} 1$ (the implication holds by confluence and the fact that 1 is a β-normal form, and for the converse take $P = M$). Then $C = \{M \in \Lambda \mid M \underset{\beta}{\twoheadrightarrow} 1\}$ would be recursive since we could use the algorithm for the pair $M, 1$. As before, $C = \{M \in \Lambda \mid M =_\beta 1\}$, which is non-recursive by Lemma 8.4.2. This is a contradiction. □

Now we move to the $\lambda\upsilon$ calculus, to which we transfer the same results. For this we recall the soundness and simulation properties of $\lambda\upsilon$ (60).

Corollary 8.4.6 *The* $\underset{\lambda\upsilon}{\twoheadrightarrow}$ *relation is undecidable.*

PROOF: Suppose it is decidable, then using the soundness property there exists an algorithm for testing $M \underset{\beta}{\twoheadrightarrow} N$ for every pair of terms $M, N \in \Lambda$. This is absurd by Corollary 8.4.4. □

To state the undecidability of $=_{\lambda\upsilon}$ we first need this

Lemma 8.4.7 *Let* $M, N \in \Lambda$*. Then* $M =_{\lambda\upsilon} N$ *iff* $M =_\beta N$

PROOF: If $M =_\beta N$, then by confluence of λ-calculus there exists a term $U \in \Lambda$ such that $M \underset{\beta}{\twoheadrightarrow} U$ and $N \underset{\beta}{\twoheadrightarrow} U$. Then by simulation, $M \underset{\lambda\upsilon}{\twoheadrightarrow} U$ and $N \underset{\lambda\upsilon}{\twoheadrightarrow} U$, thus $M =_{\lambda\upsilon} N$. For the other implication, if $M =_{\lambda\upsilon} N$, then by confluence of $\lambda\upsilon$ there exists a term $U' \in \Lambda_\upsilon^t$ such that $M \underset{\lambda\upsilon}{\twoheadrightarrow} U'$ and $N \underset{\lambda\upsilon}{\twoheadrightarrow} U'$. U' might be non pure, thus take $U = \upsilon(U')$ the υ-normal form of U', then we have $M \underset{\lambda\upsilon}{\twoheadrightarrow} U$ and $N \underset{\lambda\upsilon}{\twoheadrightarrow} U$ and then by soundness $M \underset{\beta}{\twoheadrightarrow} U$ and $N \underset{\beta}{\twoheadrightarrow} U$ thus $M =_\beta N$. □

As an application of the above we have

Corollary 8.4.8 *The $=_{\lambda\upsilon}$ relation is undecidable.*

PROOF: Suppose it is decidable, then by the previous Lemma there would exist an algorithm for testing $M =_\beta N$ for every pair $M, N \in \Lambda$, which is absurd by Corollary 8.4.3. □

We also have:

Corollary 8.4.9 *Given $M, N \in \Lambda$, the problem of deciding if there exists $P \in \Lambda$ such that $P \underset{\lambda\upsilon}{\twoheadrightarrow} M$ and $P \underset{\lambda\upsilon}{\twoheadrightarrow} N$ is undecidable.*

PROOF: $\lambda\upsilon$ satisfies soundness and simulation of the β-reduction (13), then if this problem were decidable, it would contradict Corollary 8.4.5. □

We give below a variation of the above Corollary to deal in $\lambda\upsilon$ with the existence of a common expansion to a pure term given two arbitrary Λ_υ^t terms:

Corollary 8.4.10 *Given $M, N \in \Lambda_\upsilon^t$, the problem of deciding if there exists $P \in \Lambda$ such that $P \underset{\lambda\upsilon}{\twoheadrightarrow} M$ and $P \underset{\lambda\upsilon}{\twoheadrightarrow} N$ is undecidable.*

PROOF: We reduce this problem to the one of Corollary 8.4.9. If there were an algorithm for deciding the existence of such a term, it could be used to decide the former since $\Lambda \subseteq \Lambda_\upsilon^t$. □

We prepare for the key property of this section, which is Lemma 8.4.18, stating an invariance stronger than the one handled with good contexts, and then connecting it with an undecidable problem. The goal can be seen as the necessity of taking $M = N$ in Corollary 8.4.10 and still ask if the problem remains undecidable. I.e., given a term, whether there exists a pure term reducing to it.

The following technical lemmata will be needed, where we will refer to reductions at given context positions in the usual sense.

We first show this subtle lemma relating expansion in $\lambda\upsilon$ with expansion in λ-calculus.

Lemma 8.4.11 *Let $M, N \in \Lambda$. Then there exists $P \in \Lambda_\upsilon^t$ such that $P \underset{\lambda\upsilon}{\twoheadrightarrow} M$ and $P \underset{\lambda\upsilon}{\twoheadrightarrow} N$ iff there exists $P \in \Lambda$ such that $P \underset{\beta}{\twoheadrightarrow} M$ and $P \underset{\beta}{\twoheadrightarrow} N$*

PROOF: The ($\Leftarrow$) implication is obvious by the simulation of β-reduction and because $\Lambda \subseteq \Lambda_\upsilon^t$. To check the ($\Rightarrow$) implication, we use the Projection Lemma (60). Suppose $P' \in \Lambda_\upsilon^t$, $P' \underset{\lambda\upsilon}{\twoheadrightarrow} M$ and $P' \underset{\lambda\upsilon}{\twoheadrightarrow} N$. Then take $P = \upsilon(P')$, thus by the Projection Lemma $P \underset{\beta}{\twoheadrightarrow} \upsilon(M) = M$ and $P \underset{\beta}{\twoheadrightarrow} \upsilon(N) = N$. □

Here we strengthen the notion of good context.

Definition 8.4.12 *A context $C = C\{\Box\}$ is* of closure-type *if $C\{\Box\} = D\{(D'\{\Box\})[s]\}$ with D, D' contexts and $s \in \Lambda_\upsilon^s$.*

A context $C = C\{\Box\}$ is of application-type *if $C\{\Box\} = D\{(D'\{\Box\})M\}$ with D, D' contexts and $M \in \Lambda_\upsilon^t$.*

A context $C = C\{\Box\}$ is of $\Uparrow$-type *if $C\{\Box\} = D\{M[\Uparrow (D'\{\Box\})]\}$ with D a term context, D' any context and $M \in \Lambda_\upsilon^t$. Equivalently, if there exists $k \geq 1$ such that $C\{\Box\} = D\{M[\Uparrow^k (D'\{\Box\}/)]\}$ with D, D' term contexts and $M \in \Lambda_\upsilon^t$.*

A context will be called right *if*

1. *it is not of λ-type,*
2. *it is not of $\Uparrow$-type,*
3. *it is not of application-type, and*
4. *it is not of closure-type.*

In other terms, the hole is not under the scope of a λ nor $\Uparrow$, nor inside the head of a closure, nor inside the left-hand side of an application.

Remark 8.4.13 *Let $C\{\Box\}$ be a right context and $a \in \Lambda_\upsilon^t$, then:*

- *$aC\{\Box\}$ is a right context*
- *$a[C\{\Box\}/]$ is a right context*

Definition 8.4.14 *We will say that in a term $M \in \Lambda_\upsilon^t$ a position $q \in Pos(M)$* is to the right *of a position $p \in Pos(M)$ if and only if*

- *either there is a sub-term of M which has the form PQ, and p is a position of M inside P and q is a position of M inside Q*
- *or there is a sub-term of M which has the form $P[s]$, and p is a position of M inside P and q is a position of M inside s.*

Lemma 8.4.15 *Let $M \in \Lambda_\upsilon^t$, let p be the position of a sub-term P of M and let q be the position of a sub-term Q of M. Then one and only one of the following statements holds:*

1. $p = q$
2. *q is to the right of p (equivalently, $p = \alpha.1.\beta$ and $q = \alpha.2.\gamma$ for some strings α,β,γ)*
3. *p is to the right of q (equivalently, $p = \alpha.2.\beta$ and $q = \alpha.1.\gamma$ for some strings α,β,γ)*
4. *p is a proper prefix of q (equivalently, $q = p.\alpha$ for some non-empty string α)*
5. *q is a proper prefix of p (equivalently, $p = q.\alpha$ for some non-empty string α)*

PROOF: By a simple case analysis on strings. □

In the case that item 2 or 3 holds, we say that p and q are *disjoint*.

For $M \in \Lambda^t_\upsilon$, we call *term positions* those positions from $Pos(M)$ holding sub-terms of sort term (but not of sort substitution). For instance, for a term matching $M[\Uparrow (N/)]$, 1 and 2.1.1 are term positions (the sub-terms are M and N respectively), but 2 and 2.1 are not. Note that a context will be a term context iff the hole is in a term position.

Lemma 8.4.16 *If in the context C the hole has some term position to the right, then C is not right.*

PROOF: By induction on the context C.

- If $C\{\Box\} = \Box$, the lemma holds vacuously.
- If $C\{\Box\} = C'\{\Box\}b$, clearly C is not right.
- If $C\{\Box\} = aC'\{\Box\}$, by IH C' is not right (because in C' the hole also has some term position to the right), thus C is not right.
- If $C\{\Box\} = \lambda C'\{\Box\}$, clearly C is not right.
- If $C\{\Box\} = C'\{\Box\}[s]$, clearly C is not right.
- If $C\{\Box\} = a[C'\{\Box\}]$, by IH C' is not right (because in C' the hole also has some term position to the right), thus C is not right.

□

Now the technical results:

Lemma 8.4.17 (invariance of $\Uparrow -/$ in a right context) *Let C be a right context, $B, M, P \in \Lambda^t_\upsilon$, and $k \geq 1$, such that $B \to_{\lambda\upsilon} C\{M[\Uparrow^k (P/)]\}$.*

Then there exist a right context C' terms $M', P' \in \Lambda^t_\upsilon$ and $k' \geq 1$ such that $B = C'\{M'[\Uparrow^{k'} (P'/)]\}$, and such that $P' \underset{\lambda\upsilon}{\twoheadrightarrow} P$ [1].

[1] Actually one can ensure that $P' \to^{=}_{\lambda\upsilon} P$ i.e. no more than 1 step.

PROOF: By induction on the context C. Let us call $e = M[\Uparrow^k (P/)]$. We will not specify which M', P', k' would be taken when the choice is obvious having chosen C'.

- If $C\{\Box\} = \Box$, then
  - If the reduction is at the root of B, then we analyze each $\lambda\upsilon$-rule:
    - (Beta), it is impossible since $a[b/]$ does not match e because $k \geq 1$.
    - (App), it is impossible since $a[s]b[s]$ does not match e.
    - (Lam), it is impossible since $\lambda(a[\Uparrow (s)])$ does not match e.
    - (Fvar), then take $C'\{\Box\} = 1[\Box/]$ which is a right context by Remark 8.4.13, and the result follows.
    - (Rvar), it is impossible since n does not match e.
    - (FvarLift), it is impossible since 1 does not match e.
    - (RvarLift), it is impossible since $n[s][\uparrow]$ does not match e.
    - (VarShift), it is impossible since $n + 1$ does not match e.
  - If the reduction is internal in B:
    - $B = a'b \to ab = e$ with $a' \to a$, it cannot happen since ab does not match a closure.
    - $B = ab' \to ab = e$ with $b' \to b$, it cannot happen either for the same reason.
    - $B = \lambda a' \to \lambda a = e$ with $a' \to a$, it cannot happen since λa does not match a closure.
    - $B = a'[s] \to a[s] = e$ with $a' \to a$, then $a = M$ and $s =\Uparrow^k (P/)$, so take $C'\{\Box\} = \Box$ which is right.
    - $B = a[s'] \to a[s] = e$ with $s' \to s$, then $a = M$ and $s' =\Uparrow^k (P'/)$ with $P' \to P$, so take again $C'\{\Box\} = \Box$ which is right.
- If $C\{\Box\} \neq \Box$, then
  - If the reduction is at the root of B, then we analyze each $\lambda\upsilon$-rule:
    - for rule $(\lambda a)b \to_{Beta} a[b/]$, the hole should be in b since C is right, thus $b = D\{e\}$, for D a right context; take $C'\{\Box\} = (\lambda a)D\{\Box\}$ which is right by Remark 8.4.13 and the result follows.
    - for rule $(ab)[s] \to_{App} a[s]b[s]$, the hole should be in the rightmost s (otherwise the context would not be right), and then the only possibility is that there exists N such that $s = N/$ with the hole in N (otherwise, if $s =\Uparrow^d (N/)$ with $d \geq 1$, $C\{\Box\}$ would not be right), so take $C'\{\Box\} = (ab)[D\{\Box\}/]$ where D is a right context such that $N = D\{e\}$, then C' is right by Remark 8.4.13 and the result follows.

* for rule $(\lambda a)[s] \to_{Lam} \lambda(a[\Uparrow (s)])$, whatever position has the hole, C would not be right so this case is discarded
* for rule $1[a/] \to_{Fvar} a$, take $C'\{\Box\} = 1[C\{\Box\}/]$ which is right by Remark 8.4.13.
* for rule $(n+1)[a/] \to_{Rvar} n$, the result holds vacuously since n does not match $C\{e\}$.
* for rule $1[\Uparrow (s)] \to_{Rvar} 1$, the result holds vacuously since 1 does not match $C\{e\}$.
* for rule $(n+1)[\Uparrow (s)] \to_{Rvar} n[s][\uparrow]$, the hole cannot be located in n since n does not match $C\{e\}$, nor in s for C would not be right, neither the hole can be at the position of $n[s]$ since again C would not be right, thus this case is discarded.
* for rule $n[\uparrow] \to_{VarShift} n+1$, the result holds vacuously since $n+1$ does not match $C\{e\}$.

– If the reduction is internal in B:

* for $B = a'b \to_{\lambda\upsilon} ab = C\{e\}$ where $a' \to_{\lambda\upsilon} a$, then the hole must be located in b (otherwise C would not be right). Then there is a right context D such that $b = D\{e\}$. Take $C'\{\Box\} = a'D\{\Box\}$ which is right by Remark 8.4.13, and the result follows.
* for $B = ab' \to_{\lambda\upsilon} ab = C\{e\}$ where $b' \to_{\lambda\upsilon} b$, then as before the hole must be located in b (otherwise C would not be right). Then there is a right context D such that $b = D\{e\}$. By IH, there exists a right context D' such that $b' = D'\{M'[\Uparrow^{k'} (P'/)]\}$ for $M', P' \in \Lambda^t_\upsilon$ and $k' \geq 1$ with $P' \underset{\lambda\upsilon}{\twoheadrightarrow} P$. Take $C'\{\Box\} = aD'\{\Box\}$ which is right by Remark 8.4.13 and the result follows.
* for $B = \lambda a' \to_{\lambda\upsilon} \lambda a = C\{e\}$ where $a' \to_{\lambda\upsilon} a$, it cannot happen because C would not be right.
* for $B = a'[s] \to_{\lambda\upsilon} a[s] = C\{e\}$ where $a' \to_{\lambda\upsilon} a$, then the hole must be located in s (otherwise C would not be right). Then s cannot be $\Uparrow (s')$ for any substitution s' since C is right, thus $s = R/$ for some term R thus $R = D\{e\}$ for some right context D. Take $C'\{\Box\} = a'[D\{\Box\}/]$ which is right by Remark 8.4.13 and the result follows.
* for $B = a[s'] \to_{\lambda\upsilon} a[s] = C\{e\}$ where $s' \to_{\lambda\upsilon} s$, then as before the hole must be located in s (otherwise C would not be right), and s cannot be $\Uparrow (s'')$ for any substitution s'' since the hole is in s and C is right, thus $s = R/$ for some term R, and since $s' \to_{\lambda\upsilon} s$, there exists R' such that $s' = R'/$ with $R' \to_{\lambda\upsilon} R$, thus by IH $R' = D'\{M'[\Uparrow^{k'} (P'/)]\}$ with D' right and $P' \underset{\lambda\upsilon}{\twoheadrightarrow} P$, so take $C'\{\Box\} = a[D'\{\Box\}/]$ which is right by Remark 8.4.13 and the result follows.

□

Lemma 8.4.18 (invariance of $\Uparrow -/$ in an application of right contexts) *Let C be a any context, and let C_1, C_2 be right contexts, $B, M, N, P, Q \in \Lambda_\upsilon^t$, and $k, r \geq 1$, such that $B \to_{\lambda\upsilon} C\{C_1\{M[\Uparrow^k (P/)]\}C_2\{N[\Uparrow^r (Q/)]\}\}$, where the reduction is not an (App)-step at the position of $\Box$ in $C\{\Box\}$.*

In other words, we assume it is not the case that $B = C\{(ab)[s]\}$ for some terms a, b and substitution s such that $(ab)[s] \to_{App} C_1\{M[\Uparrow^k (P/)]\}C_2\{N[\Uparrow^r (Q/)]\}$ this being a reduction at the root.

Then there exist a context C', right contexts C'_1, C'_2, terms $M', N', P', Q' \in \Lambda_\upsilon^t$ and $k', r' \geq 1$ such that $B = C'\{C'_1\{M'[\Uparrow^{k'} (P'/)]\}C'_2\{N'[\Uparrow^{r'} (Q'/)]\}\}$, and such that $P' \underset{\lambda\upsilon}{\twoheadrightarrow} P$ and $Q' \underset{\lambda\upsilon}{\twoheadrightarrow} Q$[1].

Proof: By induction on the context C. Call $e = C_1\{M[\Uparrow^k (P/)]\}C_2\{N[\Uparrow^r (Q/)]\}$. As in the previous lemma, we will not give $M', N', P', Q', k', r', C'_1, C'_2$ when the choice is clear once C' has been chosen.

- If $C\{\Box\} = \Box$, then
  - If the reduction takes place at the root of B, then we analyze each $\lambda\upsilon$-rule:
    - (Beta), it is impossible since $a[b/]$ does not match an application.
    - (App), it is impossible by hypothesis: an (App)-step is not done at the root, which is the position of the hole of C.
    - (Lam), it is impossible since $\lambda(a[\Uparrow (s)])$ does not match an application.
    - (Fvar), then take $C'\{\Box\} = 1[C\{\Box\}/\}$ and the result follows.
    - (Rvar), it is impossible since n does not match an application.
    - (FvarLift), it is impossible since 1 does not match an application.
    - (RvarLift), it is impossible since $n[s][\uparrow]$ does not match an application.
    - (VarShift), it is impossible since $n + 1$ does not match an application.
  - If the reduction is internal in B:
    - case $B = a'b \to_{\lambda\upsilon} ab$, with $a' \to_{\lambda\upsilon} a$, then $ab = e$ thus $a = C_1\{M[\Uparrow^k (P/)]\}$ and $b = C_2\{N[\Uparrow^r (Q/)]\}$. By Lemma 8.4.17, there exists C'_1 a right context, M', $k' \geq 1$ and P' such that $a' = C'_1\{M'[\Uparrow^{k'} (P'/)]\}$ and $P' \underset{\lambda\upsilon}{\twoheadrightarrow} P$, thus take $C'\{\Box\} = \Box$.
    - case $B = ab' \to_{\lambda\upsilon} ab$, with $b' \to_{\lambda\upsilon} b$, then analogously $ab = e$ thus $a = C_1\{M[\Uparrow^k (P/)]\}$ and $b = C_2\{N[\Uparrow^r (Q/)]\}$. By Lemma 8.4.17, there exists C'_2 a right context, N', $r' \geq 1$ and Q' such that $b' = C'_2\{N'[\Uparrow^{r'} (Q'/)]\}$ and $Q' \underset{\lambda\upsilon}{\twoheadrightarrow} Q$, thus take $C'\{\Box\} = \Box$.
    - case $B = \lambda a' \to_{\lambda\upsilon} \lambda a$ with $a' \to_{\lambda\upsilon} a$ is discarded since λa does not match an application.

[1] Actually one can ensure that $P' \to^{=}_{\lambda\upsilon} P$ and $Q' \to^{=}_{\lambda\upsilon} Q$, i.e. 0 or 1 step for both reductions.

* case $B = a'[s] \to_{\lambda\upsilon} a[s]$ with $a' \to_{\lambda\upsilon} a$ is discarded since $a[s]$ does not match an application.
* case $B = a[s'] \to_{\lambda\upsilon} a[s]$ with $s' \to_{\lambda\upsilon} s$ is discarded since $a[s]$ does not match an application.

- If $C\{\Box\} \neq \Box$, then
  - If the reduction is at the root of B, then we analyze each $\lambda\upsilon$-rule:
    * for rule $(\lambda a)b \to_{Beta} a[b/]$, then wherever the hole is located, the result follows. For example, if the hole is in b, then $b = D\{e\}$, for D a context; take $C'\{\Box\} = (\lambda a)D\{\Box\}$ and the result follows.
    * for rule $(ab)[s] \to_{App} a[s]b[s]$, wherever the hole is located, the result follows for an appropriate context C', namely:
      1. if the hole is in the first s, take $C'\{\Box\} = (ab)[D\{\Box\}]$ where $s = D\{e\}$
      2. if the hole is in the second s, take C' as above
      3. if the hole is in a, take $C'\{\Box\} = (D\{\Box\}b)[s]$ where $a = D\{e\}$
      4. if the hole is in b, take $C'\{\Box\} = (aD\{\Box\})[s]$ where $b = D\{e\}$
      5. the hole cannot be at the position of $a[s]$ since $a[s]$ does not match an application
      6. the hole cannot be at the position of $b[s]$ since $b[s]$ does not match an application either
      7. the hole cannot be at the position of $a[s]b[s]$ otherwise $C\{\Box\} = \Box$ which corresponds to a previously analyzed case
    * for rule $(\lambda a)[s] \to_{Lam} \lambda(a[\Uparrow (s)])$, again wherever the hole is located, the result follows for an appropriate context C'.
    * for rule $1[a/] \to_{Fvar} a$, take $C'\{\Box\} = 1[C\{\Box\}/]$.
    * for rule $(n+1)[a/] \to_{Rvar} n$, the result holds vacuously since n does not match $C\{e\}$.
    * for rule $1[\Uparrow (s)] \to_{FvarLift} 1$, the result holds vacuously since 1 does not match $C\{e\}$.
    * for rule $(n+1)[\Uparrow (s)] \to_{RvarLift} n[s][\uparrow]$, then wherever the hole is located, the result follows for an appropriate context C'.
    * for rule $n[\uparrow] \to_{VarShift} n+1$, the result holds vacuously since $n+1$ does not match $C\{e\}$.
  - If the reduction is internal in B:
    * for $B = a'b \to_{\lambda\upsilon} ab = C\{e\}$ where $a' \to_{\lambda\upsilon} a$, then we have:
      1. if the hole is located in b then there is a context D such that $b = D\{e\}$. Take $C'\{\Box\} = a'D\{\Box\}$ and the result follows.

2. if the hole is in a then there is a context D such that $a = D\{e\}$. By IH, there exists a context D', right contexts C'_1 and C'_2, terms M', N', P', Q' and $k', r' \geq 1$ such that $a' = D'\{C'_1\{M'[\Uparrow^{k'}(P'/)]\}C'_2\{N'[\Uparrow^{r'}(Q'/)]\}\}$, $P' \underset{\lambda\upsilon}{\twoheadrightarrow} P$ and $Q' \underset{\lambda\upsilon}{\twoheadrightarrow} Q$. Take $C'\{\Box\} = D'\{\Box\}b$ and the result follows.

* for $B = ab' \to_{\lambda\upsilon} ab = C\{e\}$ where $b' \to_{\lambda\upsilon} b$, it is analogous to the previous case.

* for $B = \lambda a' \to_{\lambda\upsilon} \lambda a = C\{e\}$ where $a' \to_{\lambda\upsilon} a$, then $\Box \in a$ and by IH there exists a context D such that $a' = D'\{M'[\Uparrow^{k'}(P'/)]N'[\Uparrow^{r'}(Q'/)]\}$ for $M', N', P', Q' \in \Lambda^t_\upsilon$ and $k', r' \geq 1$ with $P' \underset{\lambda\upsilon}{\twoheadrightarrow} P$ and $Q' \underset{\lambda\upsilon}{\twoheadrightarrow} Q$. Take $C'\{\Box\} = \lambda D'\{\Box\}$ and the result follows.

* for $B = a'[s] \to_{\lambda\upsilon} a[s] = C\{e\}$ where $a' \to_{\lambda\upsilon} a$, then we have:

  1. if the hole is located in s then s cannot be $\Uparrow^n(\uparrow)$ for some $n \geq 0$, so $s = \Uparrow^n(R/)$ for some term R and $n \geq 0$, thus $R = D\{e\}$ for some context D. Take $C'\{\Box\} = a'[\Uparrow^n(D\{\Box\}/)]$ and the result follows.

  2. if the hole is located in a then $a = D\{e\}$ for some context D. By IH there exist a context D', right contexts C'_1 and C'_2, terms M', N', P', Q' and $k', r' \geq 1$ such that $a' = D'\{C'_1\{M'[\Uparrow^{k'}(P'/)]\}C'_2\{N'[\Uparrow^{r'}(Q'/)]\}\}$, $P' \underset{\lambda\upsilon}{\twoheadrightarrow} P$ and $Q' \underset{\lambda\upsilon}{\twoheadrightarrow} Q$. Take $C'\{\Box\} = D'\{\Box\}[s]$ and the result follows.

* for $B = a[s'] \to_{\lambda\upsilon} a[s] = C\{e\}$ where $s' \to_{\lambda\upsilon} s$, then s cannot be $\Uparrow^n(\uparrow)$ for some $n \geq 0$, so $s = \Uparrow^n(R/)$ for some term R and $n \geq 0$, and since $s' \to_{\lambda\upsilon} s$ then there exists R' such that $s' = \Uparrow^n(R'/)$ with $R' \to_{\lambda\upsilon} R$, thus:

  1. if the hole is located in s then, since it is in a term position, it should be located in R, then by IH $R' = D'\{C'_1\{M'[\Uparrow^{k'}(P'/)]\}C'_2\{N'[\Uparrow^{r'}(Q'/)]\}\}$ with C'_1 and C'_2 right contexts, $P' \underset{\lambda\upsilon}{\twoheadrightarrow} P$ and $Q' \underset{\lambda\upsilon}{\twoheadrightarrow} Q$, so take $C'\{\Box\} = a[\Uparrow^n(D'\{\Box\}/)]$ and the result follows.

  2. if the hole is located in a then there is a context D such that $a = D\{e\}$. Take $C'\{\Box\} = D\{\Box\}[s']$ and the result follows.

□

Note that the only $\lambda\upsilon$-reduction which cannot take place is an (App)-step in the hole of the context C (but (App)-steps at other positions as well as other rules everywhere are allowed).

Actually Lemmas 8.4.17 and 8.4.18 can be stated to hold for $k = r = k' = r' = 1$ but the proof is almost the same. The case $k = 1$ is enough to prove Proposition 8.4.19 below.

We can state now a key property which relates the *common expansion* problem with the problem of expansion to a pure term:

Proposition 8.4.19 *Let $P, Q \in \Lambda$. Then the term $\lambda(1[\Uparrow(P/)]1[\Uparrow(Q/)])$ expands to a pure term iff $\exists R \in \Lambda\ R \underset{\beta}{\twoheadrightarrow} P$ and $R \underset{\beta}{\twoheadrightarrow} Q$.*

PROOF:

- ($\Leftarrow$) Suppose there exists such a term R. Then by simulation of $\lambda\upsilon$ we have that $R \underset{\lambda\upsilon}{\twoheadrightarrow} P$ and $R \underset{\lambda\upsilon}{\twoheadrightarrow} Q$, therefore

$$\begin{array}{cc} \lambda(1[\Uparrow(P/)]1[\Uparrow(Q/)]) & \\ \uparrow * & \lambda\upsilon \\ \lambda(1[\Uparrow(R/)]1[\Uparrow(R/)]) & \\ \uparrow & (App) \\ \lambda((11)[\Uparrow(R/)]) & \\ \uparrow & (Lam) \\ (\lambda 11)[R/] & \\ \uparrow & (Beta) \\ (\lambda\lambda 11)R & \end{array}$$

  which is clearly a pure term.

- ($\Rightarrow$) Let $t = \lambda(1[\Uparrow(P/)]1[\Uparrow(Q/)])$. This term clearly satisfies the conditions of Lemma 8.4.18, i.e. $t = C\{C_1\{M[\Uparrow^k(P)]\}C_2\{N[\Uparrow^r(Q)]\}\}$ where $C_1\{\Box\} = C_2\{\Box\} = \Box$, $C\{\Box\} = \lambda\Box$ and $k = r = 1$, then C_1 and C_2 are clearly right contexts.

  By hypothesis there exists a pure term B such that $B \underset{\lambda\upsilon}{\twoheadrightarrow} t$, thus there exist $t_1 \in \Lambda$, $t_2, \ldots, t_m \in \Lambda^t_\upsilon$ such that $B = t_1 \to_{\lambda\upsilon} t_2 \to_{\lambda\upsilon} \ldots \to_{\lambda\upsilon} t_m = t$.

  But since B is pure, $n_\Uparrow(B) = 0$ thus, iterating Lemma 8.4.18 it should exist $1 \leq i < m$ such that $t_i \to_{App} t_{i+1}$ where this reduction is at the position of $\Box$ in $C^{(i)}\{\Box\}$ the context corresponding to term t_i (otherwise the conclusion of the Lemma would imply that $n_\Uparrow(B) > 0$). Take $i =$ the maximum of such values, that is the first i satisfying this condition counting from the rightmost term in the derivation (i.e. t).

  This means that (within this derivation) there exist terms U, V, right contexts D, E and terms R', R'' such that

$$t_{i+1} = C^{(i+1)}\{D\{U[\Uparrow^k(R'/)]\}E\{V[\Uparrow^r(R''/)]\}\}$$

  where $R' \underset{\lambda\upsilon}{\twoheadrightarrow} P$ and $R'' \underset{\lambda\upsilon}{\twoheadrightarrow} Q$, and $C^{(i+1)} = C^{(i)}$ is the same context of the term $t_i = C^{(i)}\{(ab)[s]\}$.

  Matching the (App)-rule pattern, there are terms a, b and a substitution s such that $a[s] = D\{U[\Uparrow^k(R'/)]\}$ and $b[s] = E\{V[\Uparrow^r(R''/)]\}$.

  We will show that $R' = R''$. We reason according to D and E and, in some cases, the position of the holes with respect to the terms $a[s]$ and $b[s]$:

  - $D = \Box$ and $E = \Box$: then $\Uparrow^k(R'/) = s = \Uparrow^r(R''/)$ thus $U = V$, $k = r$ and $R' = R''$.

- $D = \Box$ and $E \neq \Box$: $a[s] = U[\Uparrow^k (R'/)]$, then $a = U$ and $s = \Uparrow^k (R'/)$. Since the hole of E is in $b[s] = E\{V[\Uparrow^r (R''/)]\}$, this hole cannot be located in b (since E is right), therefore it should be in $s = \Uparrow^k (R'/)$ and then in R'. Since $k \geq 1$, this contradicts the fact that E is right. So this will not be the case.
- $D \neq \Box$ and $E = \Box$ is analogous to the previous case: $b[s] = V[\Uparrow^r (R''/)]$, then $b = V$ and $s = \Uparrow^r (R''/)$. Since the hole of D is in $a[s] = D\{U[\Uparrow^k (R'/)]\}$, this hole cannot be located in a (since D is right), therefore it should be in $s = \Uparrow^r (R''/)$ and then in R''. Since $r \geq 1$, this contradicts the fact that D is right. So this will not be the case either.
- $D \neq \Box$ and $E \neq \Box$, and either the hole of D is located inside a or the hole of E is located inside b, but this cannot happen because both D and E are right contexts, so this case is also discarded.
- $D \neq \Box$ and $E \neq \Box$, and the hole of D is located inside the closure of $a[s]$ and the hole of E is located inside the closure of $b[s]$, in other words both holes are located inside the substitution s. Recall $a[s] = D\{U[\Uparrow^k (R'/)]\}$ and $b[s] = E\{V[\Uparrow^r (R''/)]\}$. We will show that the position of these two holes need to be the same. According to Lemma 8.4.15 the following cases can occur:
  * either both holes are in exactly the same position in s, then we are done
  * or both holes are in disjoint positions in s, then the position of one of the holes has a term position to the right (the position of the other hole), therefore by Lemma 8.4.16 the corresponding context would not be right, which is absurd.
  * or the position of E's hole is a proper prefix of the position of D's hole, i.e. D's hole is located inside E's hole position. This means that D's hole is inside the sub-term $V[\Uparrow^r (R''/)]$, then there are three cases to check:
    1. the position of D's hole is the position of $V[\Uparrow^r (R''/)]$, but this will not be the case since the prefix was supposed to be proper (in other terms, this case corresponds to a previously analyzed one in which both holes' positions coincide)
    2. the position of D's hole is inside V, but this is a closured term and then D would not be right, so this cannot happen
    3. the position of D's hole is inside $\Uparrow^r (R''/)$. Since D is a term context, the hole's position needs to be in R''. This position is under the scope of a $\Uparrow$ (since $r \geq 1$), which is absurd since D is right.
  * or the position of D's hole is a proper prefix of the position of E's hole, i.e. E's hole is located inside D's hole position. This case is analogous to the previous one exchanging the roles of D and E.

Then the only possibility is that the two holes have the same position in s. This entails $U = V$, $k = r$ and $R' = R''$.

Now we have that $R' = R''$, $R' \underset{\lambda\upsilon}{\twoheadrightarrow} P$ and $R' \underset{\lambda\upsilon}{\twoheadrightarrow} Q$. Then, since P and Q are pure, by Lemma 8.4.11 there exists a pure term R such that $R \underset{\beta}{\twoheadrightarrow} P$ and $R \underset{\beta}{\twoheadrightarrow} Q$.

□

Remark that the context C in Lemma 8.4.18 is just any context and does not need to have any special requirement. One of the tricks of this proof is initially taking P and Q not arbitrary terms but pure terms. Then, regardless if some of their expansions within the derivation (i.e. some previous terms R_j in the closures $\Uparrow^{k_j} (R_j/)$) are not pure, the implication still holds. Another tricky part is the fact that in each step the contexts are right, so this allows to keep track of the "terms under slash inside the closures" from one term to the previous one, and this enables to build a derivation from the pure term R to the two arbitrary terms P and Q initially taken.

We now state the main result of this section:

Corollary 8.4.20 *The set Λ^p_υ is recursively enumerable but non-recursive. Thus, there is no algorithm for deciding, given a term $M \in \Lambda^t_\upsilon$, whether it expands to a pure term.*

Proof: It is clearly recursively enumerable since a non-terminating algorithm could, from syntax, enumerate all pure terms and their reducts by systematically applying all reduction steps from these terms.

If it were recursive, by Proposition 8.4.19 we could test, given terms $M, N \in \Lambda$, whether there exists $P \in \Lambda$ such that $P \underset{\beta}{\twoheadrightarrow} M$ and $P \underset{\beta}{\twoheadrightarrow} N$, which is undecidable by Corollary 8.4.5. □

One of our initial goals when studying terms with pure expansion was to provide inference rules which allow to characterize the syntax of these terms, specially context-free rules in the usual way. In other terms, to have a syntax for Λ^p_υ, the $\lambda\upsilon^p$ term set, in the same way that we have a syntax for $\lambda\upsilon$-terms. We now prove that this is not possible.

Corollary 8.4.21 *The set Λ^p_υ is not a context-sensitive (type 1) language (in particular, not a context-free language), i.e. there is no context-sensitive grammar generating those terms.*

Proof: All context-sensitive languages are recursive, hence by Corollary 8.4.20 Λ^p_υ is not context-sensitive. □

8.4.1 Discussion

We point that the proof cannot be simplified by using the argument of Scott's theorem applied to $\lambda\upsilon$. In other words, even if one could prove a version of this theorem for $\lambda\upsilon$ (and for most other good calculi), it is not plausible to use it for showing that Λ^p_υ is non-recursive, as justified next. Scott's theorem for $\lambda\upsilon$ can be nicely formulated as follows:

Proposition 8.4.22 *Let $C \subseteq \Lambda^t_\upsilon$ such that $\emptyset \neq C \cap \Lambda \neq \Lambda$ and C is closed under $=_{\lambda\upsilon}$. Then C is non-recursive. (Note: the condition $C \cap \Lambda \neq \Lambda$ can be replaced by $C \neq \Lambda^t_\upsilon$, which with the other conditions implies the former)*

Proof: Take $D = C \cap \Lambda$. We prove that D is closed under $=_\beta$. Let $a \in D$ and $a =_\beta b$, then by Lemma 8.4.7 $a =_{\lambda\upsilon} b$, thus, since C is closed under $=_{\lambda\upsilon}$, $b \in C$, and since $b \in \Lambda$, $b \in D$. Now that D is closed under $=_\beta$, the hypothesis of the classical Scott's theorem is fulfilled, therefore D is non-recursive.

Last, if C were recursive, since Λ is recursive their intersection D would be recursive, contradicting the above statement. Therefore, C is non-recursive.[1] □

But then the set Λ^p_υ does not fulfill the condition of being closed under $=_{\lambda\upsilon}$ required by Proposition 8.4.22, as the following extremely simple example indicates: $1[\uparrow] \to_{\lambda\upsilon} 2$ thus $1[\uparrow] =_{\lambda\upsilon} 2$, clearly $2 \in \Lambda^p_\upsilon$ but $1[\uparrow] \notin \Lambda^p_\upsilon$ by Corollary 8.3.9.

Therefore it is not possible to show the non-recursiveness of Λ^p_υ directly from Proposition 8.4.22. This justifies the previous analysis.

8.4.2 Properties of $\lambda\upsilon^p$

We now give a straightforward characterization of the terms Λ^p_υ. By the results of the previous subsections we can only propose a pseudo-syntax for the set Λ^p_υ since it is impossible to give a context-sensitive (let alone a context-free) syntax. We will just give a finite set of inference rules based on the calculus syntax and rules altogether, describing the set of terms Λ^p_υ.

For every rule $l \to r$ in $\lambda\upsilon$, we write it as the rule

$$\frac{l}{r}$$

conforming, together with compatibility rules, an inference system as follows:

[1] Note that if instead of $\emptyset \neq C \cap \Lambda \neq \Lambda$ one requires $\upsilon(C)$ to be a proper subset of Λ^t_υ, with C closed under $=_{\lambda\upsilon}$, the argument fails when trying to prove that $\upsilon(C)$ is recursive if C were recursive.

Definition 8.4.23 (Rules describing Λ_υ^p) *Let $\mathcal{P} \subseteq \Lambda_\upsilon^t$ be the smallest set of $\lambda\upsilon$-terms closed under the following rules in which C denotes any $\lambda\upsilon$-context:*

$$\frac{m \geq 1}{m \in \mathcal{P}}\,(var) \qquad \frac{a \in \mathcal{P}}{\lambda a \in \mathcal{P}}\,(abs)$$

$$\frac{a \in \mathcal{P} \quad b \in \mathcal{P}}{ab \in \mathcal{P}}\,(app) \qquad \frac{a \in \mathcal{P} \quad a \to_{\lambda\upsilon} b}{b \in \mathcal{P}}\,(MP)$$

So with a *modus ponens*-like (MP) rule and appropriate context handling rules the set Λ_υ^p can be generated. There is an analogy with logic proofs taking as an axiom schema the (var)-rule and, as inference rule schemas, rules (abs), (app) and (MP). This is an almost straightforward inference system for Λ_υ^p, in the sense that for every term $t \in \Lambda_\upsilon^p$ there exists a derivation tree of t.

Proposition 8.4.24 $\mathcal{P} = \Lambda_\upsilon^p$.

Proof: $\mathcal{P} \subseteq \Lambda_\upsilon^p$ is proved by induction on the derivation of $M \in \mathcal{P}$.
$\Lambda_\upsilon^p \subseteq \mathcal{P}$ is proved by induction on the length of the derivation of $N \underset{\lambda\upsilon}{\twoheadrightarrow} M$ with $N \in \Lambda$. □

Corollary 8.4.25 states that $\lambda\upsilon^p$ with less terms enjoys the same good properties of its "parent" calculus $\lambda\upsilon$.

Corollary 8.4.25 (Preservation of $\lambda\upsilon$ properties) *The $\lambda\upsilon^p$ calculus satisfies the following properties:*

1. *Simulation of the β-reduction*
2. *Soundness*
3. *Confluence*
4. *PSN*
5. *Subject reduction*
6. *Expansion to pure terms*
7. *SN of typable terms*

Proof:

1. If $A \to_\beta B$ with $A, B \in \Lambda$, then $A \underset{\lambda\upsilon}{\twoheadrightarrow} B$. Then $B \in \Lambda_\upsilon^p$, so $A \underset{\lambda\upsilon^p}{\twoheadrightarrow} B$ and we are done.
2. Trivial since $\lambda\upsilon^p$ is a sub-ARS of $\lambda\upsilon$, which is sound.

3. If $A \underset{\lambda\upsilon^p}{\twoheadrightarrow} B$ and $A \underset{\lambda\upsilon^p}{\twoheadrightarrow} C$ then by confluence of $\lambda\upsilon$ there exists a $D \in \Lambda^t_\upsilon$ such that $B \underset{\lambda\upsilon}{\twoheadrightarrow} D$ and $C \underset{\lambda\upsilon}{\twoheadrightarrow} D$. Sice $A \in \Lambda^p_\upsilon$, then $B \in \Lambda^p_\upsilon$ and thus $D \in \Lambda^p_\upsilon$. In other words, $\lambda\upsilon^p$ is CR for being a subsystem of $\lambda\upsilon$.

4. From the fact that $\lambda\upsilon$ satisfies PSN.

5. This is an immediate consequence of $\lambda\upsilon^p \subset \lambda\upsilon$.

6. Immediate by definition.

7. $\lambda\upsilon^p$ admits simply-typing with the same typing rules of $\lambda\upsilon$ (60). If $M \in \Lambda^p_\upsilon$ is a simply-typed term then it is also simply-typed in Λ^t_υ, thus $M \in SN_{\lambda\upsilon}$, and therefore since $\lambda\upsilon^p \subset \lambda\upsilon$, $M \in SN_{\lambda\upsilon^p}$.

□

As it can be seen, most properties are directly inherited from $\lambda\upsilon$ using the fact that $\lambda\upsilon^p$ is a sub-calculus.

It is obvious that the $\lambda\upsilon^p$ terms set cannot be extended with any other $\lambda\upsilon$-term without losing the expansion property. Also, for being a sub-calculus, this set cannot be shortened. One of the disadvantages of this new calculus is that the term set is not closed under the sub-term relation. In other words, it is easy to find a $\lambda\upsilon$-term M such that for some sub-term N of M, $N \notin \Lambda^p_\upsilon$. Take for example $N = 1[\Uparrow (1/)]$ and $M = \lambda N$.

8.5 Expansion in λs

In this section we study the expansion problem for λs. Although in the following section we transfer the previous decidability results from $\lambda\upsilon$ to λs (and the reader may jump to it since it is independent of this section), the same technique as before can be used, with a notion of good context in λs, to show directly that in this calculus there are terms which do not expand to pure terms. This is done below, and the interested reader can find the technical proofs in the Appendix at the end of this chapter.

Definition 8.5.1 (λs-terms with pure expansion) *Given $M \in \Lambda_s$ we say that M has a pure expansion if there exists $N \in \Lambda$ such that $N \underset{\lambda s}{\twoheadrightarrow} M$. Let $\Lambda^p_s = \{M \in \Lambda_s \mid M$ has a pure expansion$\}$.*

As before, these terms form a sub-calculus of λs. So we will see that this sub-ARS is also proper.

Definition 8.5.2 *For a a term in λs we define the number of σ's in a, written $n_\sigma(a)$, in the expected way:*

$$\begin{array}{lcllcl} n_\sigma(n) & = & 0 & & & \\ n_\sigma(ab) & = & n_\sigma(a)+n_\sigma(b) & n_\sigma(\lambda a) & = & n_\sigma(a) \\ n_\sigma(a\sigma^k b) & = & n_\sigma(a)+n_\sigma(b)+1 & n_\sigma(\varphi^i_k(a)) & = & n_\sigma(a) \end{array}$$

For a a term in λs we define the number of φ's in a, written $n_\varphi(a)$, in the expected way:

$$\begin{array}{lcllcl} n_\varphi(n) & = & 0 & & & \\ n_\varphi(ab) & = & n_\varphi(a)+n_\varphi(b) & n_\varphi(\lambda a) & = & n_\varphi(a) \\ n_\varphi(a\sigma^k b) & = & n_\varphi(a)+n_\varphi(b) & = \quad n_\varphi(\varphi^i_k(a)) & = & n_\varphi(a)+1 \end{array}$$

Definition 8.5.3 *A context $C\{\Box\}$ in λs will be called* good *if it is not of the form $C\{\Box\} = D\{\lambda D'\{\Box\}\}$ nor of the form $C\{\Box\} = D\{(D'\{\Box\})\sigma^1 b\}$ with $b \in \Lambda_s$, D, D' contexts.*

As with $\lambda\upsilon$, the reason of identifying these contexts is the following

Lemma 8.5.4 (invariance of σ in good contexts) *Let C be a good context, $B, a, b \in \Lambda_s, k \geq 2$, such that $B \underset{\lambda s}{\longrightarrow} C\{a\sigma^k b\}$. Then there exists a good context C', there exist $a', b' \in \Lambda_s$, and $k' \geq 1$ such that $B = C'\{a'\sigma^{k'} b'\}$ (actually $k' = k$ or $k' = k+1$).*

Proof: See the Appendix. □

Lemma 8.5.5 (invariance of φ in good contexts) *Let C be a good context, $B, a \in \Lambda_s, i \geq 2, k \geq 0$, such that $B \underset{\lambda s}{\longrightarrow} C\{\varphi^i_k(a)\}$. Then there exists a good context C', there exist $a' \in \Lambda_s$ and $i' \geq 2$ such that*

1. *either $B = C'\{\varphi^{i'}_{k'}(a')\}$ for some $k' \geq 0$*
2. *or $B = C'\{a'\sigma^{i'} b'\}$ for some $b' \in \Lambda_s$*

Proof: See the Appendix. □

As a consequence of the previous two lemmas we have the following

Proposition 8.5.6

1. *If $B \underset{\lambda s}{\twoheadrightarrow} C\{a\sigma^n b\}$ with $n \geq 2$ and C a good context, then there exists C' a good context, and there exist $a', b' \in \Lambda_s$ and $n' \geq 2$ such that $B = C'\{a'\sigma^{n'} b'\}$.*
2. *If $B \underset{\lambda s}{\twoheadrightarrow} C\{\varphi^i_k(a)\}$ with $i \geq 2$, $k \geq 0$ and C a good context, then there exists C' a good context, and there exist $a' \in \Lambda_s$ and $i' \geq 2$ such that*
   (a) *either $B = C'\{\varphi^{i'}_{k'}(a')\}$ for some $k' \geq 0$*
   (b) *or $B = C'\{a'\sigma^{i'} b'\}$ for some $b' \in \Lambda_s$.*

PROOF: Both items are proved by induction on the length of the derivation $B \underset{\lambda s}{\twoheadrightarrow} C\{a\sigma^n b\}$ using lemmas 8.5.4 and 8.5.5. □

Corollary 8.5.7 *In λs the terms of the form $a\sigma^n b$ with $n \geq 2$ and the terms of the form $\varphi_k^i(a)$ with $i \geq 2$ and $k \geq 0$ do not expand to pure terms.*

PROOF: In Proposition 8.5.6, any expansion of such a term will have the form $B = C'\{a'\sigma^{n'}b'\}$ with $n' \geq 2$ (then $n_\sigma(B) \geq k' \geq 1$), or the form $B = C'\{\varphi_{k'}^{i'}(a')\}$ (then $n_\varphi(B) \geq 1$), in either case $B \notin \Lambda$. □

Recall Definition 8.3.11 in subsection 8.3.2. As with $\lambda\upsilon$, we have

Proposition 8.5.8 *There is no surjective good mapping from $\lambda\mathbf{x}$ to λs.*

PROOF: Analogous to Proposition 8.3.13. □

8.6 Undecidability results for λs

We now will use the translations T and S between $\lambda\upsilon$ and λs, which appear in (47) and pose the problem whether they preserve the corresponding sets of terms with pure expansion.

Definition 8.6.1 *We define the following translation $T : \Lambda_s \to \Lambda_\upsilon^t$.*

$T(n) = n$
$T(ab) = T(a)T(b)$
$T(\lambda a) = \lambda T(a)$
$T(a\sigma^n b) = T(a)[\Uparrow^{n-1}(T(b)/)]$
$T(\varphi_k^i(a)) = T(a)\underbrace{[\Uparrow^k(\uparrow)]\ldots[\Uparrow^k(\uparrow)]}_{i-1} \quad i \geq 1, k \geq 0$

Definition 8.6.2 *We define the following translation $S : \Lambda_\upsilon^t \to \Lambda_s$.*

$S(n) = n$
$S(ab) = S(a)S(b)$
$S(\lambda a) = \lambda S(a) \quad n \geq 0$
$S(a[\Uparrow^n(b/)]) = (S(a))\sigma^{n+1}(S(b)) \quad n \geq 0$
$S(a[\Uparrow^n(\uparrow)]) = \varphi_n^2(S(a)) \quad n \geq 0$

The following Lemma states that these translations are morphisms between the calculi:

Lemma 8.6.3 *1. For $a, b \in \Lambda_s$, if $a \underset{\lambda s}{\longrightarrow} b$ then $T(a) \underset{\lambda\upsilon}{\twoheadrightarrow} T(b)$.*

2. *For* $a, b \in \Lambda_\upsilon^t$, *if* $a \underset{\lambda\upsilon}{\longrightarrow} b$ *then* $S(a) \underset{\lambda s}{\twoheadrightarrow} S(b)$.

3. *For* $a, b \in \Lambda_s$, *if* $a \underset{\lambda s}{\twoheadrightarrow} b$ *then* $T(a) \underset{\lambda\upsilon}{\twoheadrightarrow} T(b)$.

4. *For* $a, b \in \Lambda_\upsilon^t$, *if* $a \underset{\lambda\upsilon}{\twoheadrightarrow} b$ *then* $S(a) \underset{\lambda s}{\twoheadrightarrow} S(b)$.

PROOF: Items (1) and (2) are proved by induction on a. Items (3) and (4) are proved by induction on the length of the derivations using items (1) and (2), respectively. For details see (47). □

Lemma 8.6.4 1. *For* $M \in \Lambda_\upsilon^t$, $T(S(M)) = M$.

2. *For* $N \in \Lambda_s$, $S(T(N)) =_s N$.

PROOF:

1. By induction on M.
   - $T(S(n)) = T(n) = n$
   - $T(S(ab)) = T(S(a)S(b)) = T(S(a))T(S(b)) =_{IH} ab$
   - $T(S(\lambda a)) = T(\lambda S(a)) = \lambda T(S(a)) =_{IH} \lambda a$
   - $T(S(a[\Uparrow^n (b/)])) = T(S(a)\sigma^{n+1}S(b))$
     $= T(S(a))[\Uparrow^{n+1-1} (T(S(b))/)] =_{IH} a[\Uparrow^n (b/)]$
   - $T(S(a[\Uparrow^n (\uparrow)])) = T(\varphi_n^2(S(a)))$
     $= T(S(a))[\Uparrow^n (\uparrow)] =_{IH} a[\Uparrow^n (\uparrow)]$
2. By induction on N (we omit it since we will not use it).

□

So S and T are not inverses of each other, but T is a pseudo-inverse of S in the sense that for all $a \in \Lambda_s$ $S(T(a)) =_s a$ but not necessarily $S(T(a)) = a$. Nevertheless this is all what we need to relate both calculi in the context of the present goal.

These morphisms between $\lambda\upsilon$ and λs will help to state the corresponding decidability result for λs-terms by translating λs-terms to $\lambda\upsilon$ and using the result from section 8.4. We must say that using this translation does not provide a translation of Lemmas 8.3.6 and 8.3.7, hence the need of Lemmas 8.5.4 and 8.5.5 for λs to explore terms not in Λ_s^p. On the other hand, it is possible to translate Corollary 8.4.20 to λs as we will see next.

Lemma 8.6.5 *The following statements are equivalent:*

1. $M \in \Lambda$

2. $M \in \Lambda_s$ *and* $T(M) = M$

3. $M \in \Lambda_v^t$ *and* $S(M) = M$

4. $M \in \Lambda_s$ *and* $T(M) \in \Lambda$

5. $M \in \Lambda_v^t$ *and* $S(M) \in \Lambda$

PROOF: Easy. (1)⇒(2) and (1)⇒(3) are proved by induction on M. □

Actually we will only need (1)⇒(2) and (1)⇒(3).

Lemmas 8.6.3 and 8.6.5 indicate that T and S are good mappings between both calculi. Now we relate the sets Λ_v^p and Λ_s^p.

Lemma 8.6.6 *Let $M \in \Lambda_v^t$. Then M λv-expands to a pure term iff $S(M)$ λs-expands to a pure term. In other words, $S(\Lambda_v^p) \subseteq \Lambda_s^p$.*

PROOF: Suppose there exists $P \in \Lambda$ such that $P \underset{\lambda v}{\twoheadrightarrow} M$. Apply Lemma 8.6.3 (2): $S(P) \underset{\lambda s}{\twoheadrightarrow} S(M)$. But by Lemma 8.6.5 ((1)⇒(3)), $S(P) = P$ and we are done.

Now suppose there exists $P \in \Lambda$ such that $P \underset{\lambda s}{\twoheadrightarrow} S(M)$. Apply Lemma 8.6.3 (1): $T(P) \underset{\lambda v}{\twoheadrightarrow} T(S(M))$. By Lemma 8.6.5 ((1)⇒(2)) $T(P) = P$, and by Lemma 8.6.4 (1) $T(S(M)) = M$, so $P \underset{\lambda v}{\twoheadrightarrow} M$ and we are done. □

Then we have the following

Corollary 8.6.7 *The set Λ_s^p is recursively enumerable but non-recursive.*

PROOF: It is recursively enumerable by the same argument used in Corollary 8.4.20 now applied to λs.

Suppose there is an algorithm to decide, given $N \in \Lambda_s$, whether $N \in \Lambda_s^p$. We show that we could decide, given $M \in \Lambda_v^t$, whether $M \in \Lambda_v^p$. Take $N = S(M)$. Decide if $\exists P \in \Lambda\ P \underset{\lambda s}{\twoheadrightarrow} S(M)$. If so, return yes. Else return no. By Lemma 8.6.6, this decides if $\exists P \in \Lambda\ P \underset{\lambda v}{\twoheadrightarrow} M$, which is a contradiction by Corollary 8.4.20. □

Corollary 8.6.8 *The set Λ_s^p is not a context-sensitive language.*

PROOF: Same argument of Corollary 8.4.21. □

A description of Λ_s^p can be given in an analogous way to Definition 8.4.23 just replacing λv by λs.

8.7 Conclusion

In this chapter we have introduced the study of expansion in explicit substitution calculi pointing out its relevance. This is inspired in the convenience of dealing with a calculus with a minimal set of terms, leaving out the unneeded ones. As far as (Beta)-expansion and normalization is concerned, the only needed terms in a calculus are those which expand to pure terms. This leads to a smaller calculus. As an application, we showed the non-existence of certain good mappings between calculi.

It turns out that the set of terms which expand to pure terms behaves very differently in some calculus with respect to another. We get for $\lambda\upsilon$ and λs undecidability results about this set, contrarily to what happens in $\lambda\mathtt{x}$, a calculus in which not only this set is recursive but it is total, i.e. every term in $\lambda\mathtt{x}$ has a pure expansion.

What is also interesting is the fact that we had to use explicit substitutions to *code* in a term the problem of deciding *common expansion*, which also resulted undecidable and was the connection with expansion to pure terms in virtue of the (App)-rule effect.

For transferring undecidability from $\lambda\upsilon$ to λs we used the fact that $T(S(M)) = M$ (T is surjective). If we had done it the opposite way, from λs to $\lambda\upsilon$, we would have had a problem since S is not surjective.

Until now all standard variations of λ-calculus were introduced by using sets of terms that can be generated by context-free grammars or near formalisms. We have shown that not always this should be the case sometimes, and the sets of terms need to be described by other means.

We have found two kinds of contexts when looking for invariants: good contexts and right contexts, the latter being a particular case of the former. But using the former it was possible to prove that not every term has a pure expansion in $\lambda\upsilon$ and λs.

One useful consequence of the results in this chapter is the fact that in $\lambda\upsilon$ the standard set of terms Λ_υ^t seems to have the following two properties: 1) being minimal with respect to being closed under reduction rules, and 2) recursive. Property (1) was perhaps one of the main motivations for $\lambda\upsilon$ to be proposed. Property (2) seems a reasonable property. We gained more evidence of the minimality of $\lambda\upsilon$. "In other terms", if the set of terms is shortened, one would have the (somehow unpleasant) fact that this set is non-recursive. The same happens with λs.

It remains an open problem to give a method for calculating in certain situations those pure expansions when they exist. Even when there might be a "semi-algorithm" to solve this, it is not clear how to do it optimally, nor how preferable can one pure term be with respect to another.

As another facet of research, we expect to find a more accurate inference system for Λ_υ^p and Λ_s^p, possibly based on sequents. Because all these rules would be of induction nature (premises -possibly empty- and conclusion), such a description would be useful for proving properties

inductively for this set of terms, as done in (4; 82) as well as in chapter 4 for the sets of strongly normalizing terms.

In the future we expect to analyze the situation in $\lambda\sigma$, $\lambda\mathtt{w}_s$, λs_e and $\lambda\omega_e$. All of them seem non-trivial. The solution of these problems could lead to different presentations of the calculi. We did not investigate the existence of an explicit substitution calculus with de Bruijn indices with the good properties of $\lambda\upsilon$ and such that all terms which expand to pure terms form a recursive set, or better, a context-free set. This is also left for future research.

8.8 Appendix. Proofs for λs.

We give here detailed proofs of the invariance of σ and φ in good contexts for λs.

PROOF: (of Lemma 8.5.4) By induction on the context C. Call $e = a\sigma^k b$. We will not give a', b', k' when the choice is clear once C' has been chosen.

- if $C\{\Box\} = \Box$, then we have:
  - if the reduction is at the root of B:
    - if the reduction is a $(\sigma - gen)$-step, it would imply $k = 1$ but this is not possible by hypothesis.
    - it cannot be a $(\sigma - app)$-step since e does not match an application.
    - it cannot be a $(\sigma - \lambda)$-step since e does not match an abstraction.
    - for the $(\sigma - des)$-step $m\sigma^n b \to m$ with $m < n$, it cannot happen since e does not match the reduct.
    - for the $(\sigma - des)$-step $m\sigma^n b \to \varphi_0^n(b)$ with $m = n$, it cannot happen since e does not match the reduct.
    - for a $(\sigma - des)$-step $m\sigma^n b \to m - 1$ with $m > n$, it cannot happen since e does not match the reduct.
    - for a $(\varphi - app)$-step $\varphi_j^i(ab) \to \varphi_j^i(a)\varphi_j^i(b)$, it cannot happen since e does not match the reduct.
    - it cannot be a $(\varphi - \lambda)$-step since e does not match an abstraction.
    - it cannot be a $(\varphi - des)$-step since e does not match an index.
  - if the reduction is internal in B:
    - $a'b \to ab = e$ with $a' \to a$, it cannot happen since e does not match an application.
    - $ab' \to ab = e$ with $b' \to b$, it cannot happen since e does not match an application.
    - $\lambda a' \to \lambda a = e$ with $a' \to a$, it cannot happen since e does not match an abstraction.

* $a'\sigma^k b \to a\sigma^k b = e$ with $a' \to a$, immediate taking $C'\{\Box\} = \Box$ which is a good context.
* $a\sigma^k b' \to a\sigma^k b = e$ with $b' \to b$, immediate taking $C'\{\Box\} = \Box$ which is a good context.
* $\varphi^i_k(a') \to \varphi^i_k(a) = e$ with $a' \to a$, it cannot happen since e does not match the reduct.

- if $C \neq \Box$ then:
  - if the reduction is at the root of B:
    * $B = (\lambda a)b \to a\sigma^1 b$
      · if $\Box \in b$, take $C'\{\Box\} = (\lambda a)D\{\Box\}$ where $b = D\{e\}$, then C' is good because D is good.
      · if $\Box \in a$, it cannot happen since C is good.
    * $B = (ab)\sigma^n c \to (a\sigma^n c)(b\sigma^n c)$, we have the following cases:
      · if $\Box \in a$ with $n \geq 2$, take $C'\{\Box\} = (D\{\Box\}b)\sigma^n c$ where $a = D\{e\}$ with D good, thus C' is good.
      · if $\Box \in b$ with $n \geq 2$, take $C'\{\Box\} = (aD\{\Box\})\sigma^n c$ where $b = D\{e\}$ with D good, thus C' is good.
      · if $\Box \in c$ at the left (i.e. its position is of the form $1.2.q$ for some q), then take $C'\{\Box\} = (a\sigma^n(D\{\Box\}))(b\sigma^n c)$ where $c = D\{e\}$ with D good, thus C' is good.
      · if $\Box \in c$ at the right (i.e. its position is of the form $2.2.q$ for some q), then take $C'\{\Box\} = (a\sigma^n c)(b\sigma^n(D\{\Box\}))$ where $c = D\{e\}$ with D good, thus C' is good.
      · if $\Box$ occurs at the root of $a\sigma^n c = e$, take $C'\{\Box\} = D\{\Box(b\sigma^n c)\}$ which is good because D is good.
      · if $\Box$ occurs at the root of $b\sigma^n c = e$, take $C'\{\Box\} = D\{(a\sigma^n c)\Box\}$ which is good because D is good.
    * $B = (\lambda a)\sigma^n c \to \lambda(a\sigma^{n+1}c)$
      · then $\Box \in a\sigma^{n+1}c$, but this is absurd because by hypothesis C is good.
    * $B = m\sigma^n b \to m - 1$ with $m > n$, it cannot happen since $m - 1 \neq C\{e\}$.
    * $B = m\sigma^n b \to \varphi^n_0(b)$ with $m = n$, then $\Box \in b$ thus take $C'\{\Box\} = m\sigma^n D\{\Box\}$ where $b = D\{e\}$ and the result holds
    * $B = m\sigma^n b \to m$ with $m < n$, it cannot happen since $m \neq C\{e\}$.
    * $B = \varphi^i_j(ab) \to \varphi^i_j(a)\varphi^i_j(b)$
      · if $\Box \in a$, take $C'\{\Box\} = \varphi^i_j(D\{\Box\}b)$ where $a = D\{e\}$ thus C' is good because D is good.

· if $\Box \in b$, take $C'\{\Box\} = \varphi^i_j(aD\{\Box\})$ where $b = D\{e\}$ thus C' is good because D is good.

· if $\Box$ occurs at the root of $\varphi^i_j(a)$, absurd since e does not match $\varphi^i_j(a)$.

· if $\Box$ occurs at the root of $\varphi^i_j(b)$, absurd since e does not match $\varphi^i_j(b)$.

∗ $B = \varphi^i_j(\lambda a) \to \lambda\varphi^i_{j+1}(a)$

· then $\Box \in \varphi^i_{j+1}(a)$, absurd since C is good.

∗ $B = \varphi^i_j(m) \to m$, it cannot happen since $C\{e\}$ is not an index.

∗ $B = \varphi^i_j(m) \to m + i - 1$, it cannot happen $C\{e\}$ is not an index.

– If the reduction is internal in B:

∗ $B = a'b \to ab$ with $a' \to a$.

· if $\Box \in b$, take $C'\{\Box\} = a'D\{\Box\}$ which is good, where $b = D\{e\}$.

· if $\Box \in a$, by IH $a' = D'\{a'\sigma^{k'}b'\}$ with D' good, take $C'\{\Box\} = D'\{\Box\}b$ which is good.

∗ $B = ab' \to ab$ with $b' \to b$, it is analogous to the previous case.

∗ $B = \lambda a' \to \lambda a = C\{e\}$, it cannot happen because C is good.

∗ $B = a'\sigma^n b \to a\sigma^n b$ with $a' \to a$.

· if $\Box \in b$, take $C'\{\Box\} = a'\sigma^n D\{\Box\}$ where $b = D\{e\}$, then C' is good because D is good.

· if $\Box \in a$ and $n = 1$ we have an absurd because C is good.

· if $\Box \in a$ and $n \geq 2$, then since $a' \to a$, by IH $a' = D'\{a''\sigma^{n'}b'\}$ with D' good and $n' \geq 2$, thus take $C'\{\Box\} = D'\{\Box\}\sigma^n b$ which is good.

∗ $B = a\sigma^n b' \to a\sigma^n b$ with $b' \to b$.

· if $\Box \in b$, by IH $b' = D'\{a'\sigma^{n'}b''\}$ with D' good and $n' \geq 2$, thus take $C'\{\Box\} = a\sigma^n D'\{\Box\}$ which is good.

· if $\Box \in a$ and $n = 1$ we have an absurd because C is good.

· if $\Box \in a$ and $n \geq 2$, take $C'\{\Box\} = D\{\Box\}\sigma^n b'$ where $a = D\{e\}$, then C' is good because D is good.

∗ $B = \varphi^i_j(a') \to \varphi^i_j(a)$ with $a' \to a$ and $\Box \in a$, then by IH $a' = D'\{a''\sigma^{n'}b'\}$ with D' good, thus take $C'\{\Box\} = \varphi^i_j(D'\{\Box\})$ which is good.

□

PROOF: (of Lemma 8.5.5) By induction on the context C. Call $e = \varphi^i_k(a)$. Again, we will not give a', i', k', b' when the choice is clear once C' has been chosen.

- if $C\{\Box\} = \Box$, then we have:

- if the reduction is at the root of B:
  - if the reduction is a $(\sigma - gen)$-step, it would imply $e = a\sigma^1 b$ for some terms a, b, which is not possible.
  - it cannot be a $(\sigma - app)$-step since e is not an application.
  - it cannot be a $(\sigma - \lambda)$-step since e is not abstraction.
  - it cannot be a $(\sigma - des)$-step $m\sigma^n b \to m$ with $m < n$ since e is not an index.
  - if it is a $(\sigma - des)$-step $m\sigma^n b \to \varphi_0^n(b)$ with $m = n$, it would imply that $n \geq 2$, thus the result follows taking $C\{\Box\} = \Box$ which is good.
  - it cannot be a $(\sigma - des)$-step $m\sigma^n b \to m - 1$ with $m > n$ since e is not an index.
  - it cannot be a $(\varphi - app)$-step $\varphi_j^l(ab) \to \varphi_j^l(a)\varphi_j^l(b)$ since e is not an application.
  - it cannot be a $(\varphi - \lambda)$-step since e does not match an abstraction.
  - it cannot be a $(\varphi - des)$-step since e is not an index.
- if the reduction is internal in B:
  - $a'b \to ab = e$ with $a' \to a$, it cannot happen since e does not match an application.
  - $ab' \to ab = e$ with $b' \to b$, it cannot happen since e does not match an application.
  - $\lambda a' \to \lambda a = e$ with $a' \to a$, it cannot happen since e does not match an abstraction.
  - $a'\sigma^k b \to a\sigma^k b = e$ with $a' \to a$, it cannot happen since e does not match the reduct.
  - $a\sigma^k b' \to a\sigma^k b = e$ with $b' \to b$, it cannot happen since e does not match the reduct.
  - $\varphi_j^l(a') \to \varphi_j^l(a) = e$ with $a' \to a$, then by IH $a' = D'\{a''\sigma^{i'}b'\}$ or $a' = D'\{\varphi_{k'}^{i'}(a'')\}$ where D' is good, then take $C'\{\Box\} = \varphi_j^l(D'\{\Box\})$ which is good and the other conditions hold.

- if $C \neq \Box$ then
  - if the reduction is at the root of B:
    - $B = (\lambda a)b \to a\sigma^1 b$
      - if $\Box \in b$, take $C'\{\Box\} = (\lambda a)D\{\Box\}$ where $b = D\{e\}$, then C' is good because D is good.
      - if $\Box \in a$, it cannot happen since C is good.
    - $B = (ab)\sigma^n c \to (a\sigma^n c)(b\sigma^n c)$
      - the hole cannot occur at the root of $(a\sigma^n c)$ nor at the root of $(b\sigma^n c)$ since e does not match neither of them.
      - if $\Box \in a$ then $n \geq 2$ (or else C would not be good), thus take $C'\{\Box\} = (D\{\Box\}b)\sigma^n c$ where $a = D\{e\}$, then C' is good because D is good.

· if $\Box \in b$ then analogously $n \geq 2$ (or else C would not be good), thus take $C'\{\Box\} = (aD\{\Box\})\sigma^n c$ where $b = D\{e\}$, then C' is good because D is good.

· if $\Box \in c$ (either in the left-hand side or in the right-hand side of the application), take $C'\{\Box\} = (ab)\sigma^n D\{\Box\}$ where $c = D\{e\}$, then C' is good because D is good.

* $B = (\lambda a)\sigma^n c \to \lambda(a\sigma^{n+1}c)$

· then $\Box \in a\sigma^{n+1}c$, absurd because by hypothesis C is good.

* $B = m\sigma^n b \to m-1$ with $m > n$, it cannot happen since $C\{e\}$ is not an index.

* $B = m\sigma^n b \to \varphi_0^n(b)$ with $m = n$, then $\Box \in b$ thus take $C'\{\Box\} = m\sigma^n(D\{\Box\})$ where $b = D\{e\}$, then C' is good because D is good.

* $B = m\sigma^n b \to m$ with $m < n$, it cannot happen since $C\{e\}$ is not an index.

* $B = \varphi_j^l(ab) \to \varphi_j^l(a)\varphi_j^l(b)$

· if $\Box \in a$, take $C'\{\Box\} = \varphi_j^l(D\{\Box\}b)$ where $a = D\{e\}$ thus C' is good because D is good.

· if $\Box \in b$, take $C'\{\Box\} = \varphi_j^l(aD\{\Box\})$ where $b = D\{e\}$ thus C' is good because D is good.

· if $\Box$ occurs at the root of $\varphi_j^l(a)$, take $C'\{\Box\} = \Box$ which is good.

· if $\Box$ occurs at the root of $\varphi_j^l(b)$, take $C'\{\Box\} = \Box$ which is good.

* $B = \varphi_j^l(\lambda a) \to \lambda\varphi_{j+1}^l(a)$

· then $\Box \in \varphi_{j+1}^l(a)$, but this is absurd since C is good.

* $B = \varphi_j^l(m) \to m$, it cannot happen since $C\{e\}$ is not an index.

* $B = \varphi_j^l(m) \to m + i - 1$, it cannot happen since $C\{e\}$ is not an index.

– If the reduction is internal in B:

* $B = a'b \to ab$ with $a' \to a$:

· if $\Box \in b$, take $C'\{\Box\} = a'D\{\Box\}$ where $b = D\{e\}$, then C' is good because D is good.

· if $\Box \in a$, then by IH $a' = D'\{a''\sigma^{i'}b'\}$ or $a' = D'\{\varphi_{k'}^{i'}(a'')\}$ where D' is good, thus take $C'\{\Box\} = D'\{\Box\}b$ which is good and the other conditions hold.

* $B = ab' \to ab$ with $b' \to b$, it is analogous to the previous case.

* $B = \lambda a' \to \lambda a = C\{e\}$, it cannot happen because C is good.

* $B = a'\sigma^n b \to a\sigma^n b$ with $a' \to a$.

· if $\Box \in b$, take $C'\{\Box\} = a'\sigma^n D\{\Box\}$ where $b = D\{e\}$, then C' is good because D is good.

· if $\Box \in a$ and $n = 1$ we have an absurd because C is good.

- if $\Box \in a$ and $n \geq 2$, then since $a' \to a$, by IH $a' = D'\{a''\sigma^{i'}b'\}$ or $a' = D'\{\varphi^{i'}_{k'}(a'')\}$ where D' is good, thus take $C'\{\Box\} = D'\{\Box\}\sigma^n b$ which is good and the other conditions hold.

* $B = a\sigma^n b' \to a\sigma^n b$ with $b' \to b$.
  - if $\Box \in b$, then by IH $a' = D'\{a''\sigma^{i'}b'\}$ or $a' = D'\{\varphi^{i'}_{k'}(a'')\}$ where D' is good, thus take $C'\{\Box\} = a\sigma^n(D'\{\Box\})$ which is good and the other conditions hold.
  - if $\Box \in a$ and $n = 1$ we have an absurd because C is good.
  - if $\Box \in a$ and $n \geq 2$, take $C'\{\Box\} = D\{a\}\sigma^n b'$, then C' is good because D is good.
* $B = \varphi^l_j(a') \to \varphi^l_j(a)$ with $a' \to a$ and $\Box \in a$, then by IH $a' = D'\{a''\sigma^{i'}b'\}$ or $a' = D'\{\varphi^{i'}_{k'}(a'')\}$ where D' is good, thus take $C'\{\Box\} = \varphi^l_j(D'\{\Box\})$ which is good and the other conditions hold.

$\Box$

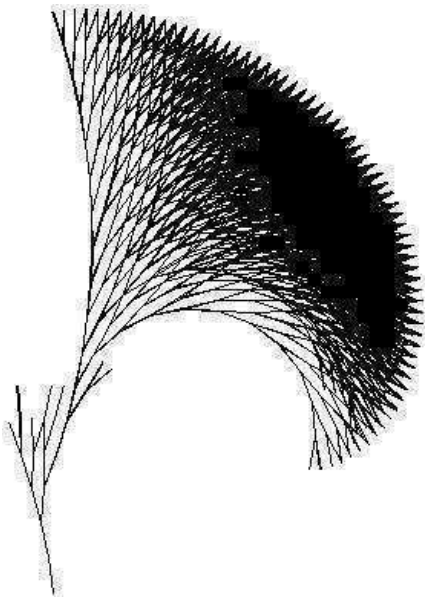
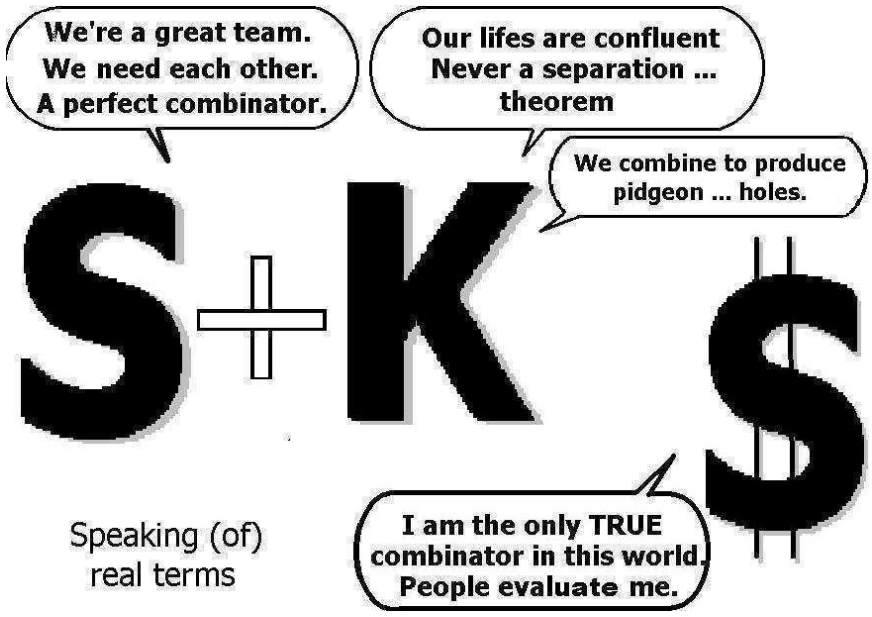


Figure 8.1: $L(LX)(L(LX)(LL))$ after 23 left-most steps



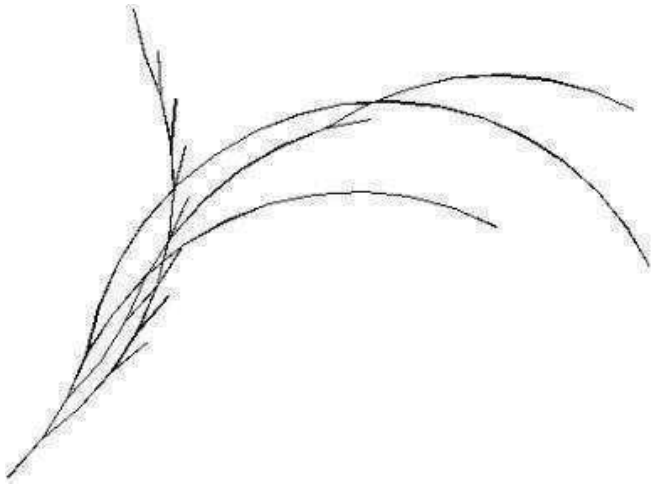


Figure 8.2: $SSSSSSSSSSSSSS(SSSSSSSSSSSSSSSS)$ after 100 left-most steps

Chapter 9

Conclusion

False convictions are much more dangerous than lies. – F. Nietzsche

Being aware of one's ignorance is a big step to wisdom. – Disraeli

Trivial problems are quickly eliminated. Important problems are never solved. – Murphy

Rewriting is a meta-level approach of computing. As we observed at the beginning of this thesis, rewriting can be seen as a formulation of, but not precisely a paradigm for, computing. It can be used in other theories, it can help to implement interpreters, compilers and translators, it can give an abstract fundament of formal language theory. It serves as a generalization of these study fields. Even other fields which seem to be so distant could benefit from rewriting, such as natural language processing, computational biology, image processing and artificial intelligence in general, due to the fact that rewriting ideas are general and basic enough.

Substitution was a concept studied in a limited way until the emergence of explicit substitution. We have noticed that different forms of minimality to the syntax can be attempted for some calculi, for example the number of rules, the term set, and the number of syntactical operators.

In this thesis we have traveled over a bunch of aspects and motivations of rewriting theory, according to the concepts from the literature. Our goal was to consider problems which relate to rewriting systems and some of their subsystems. Subsystems appear in different forms: considering subsets of terms, considering subsets of rules - in general, considering special (usual) rewriting-related properties, and restricting the rule applications.

We discussed well-known issues of rewriting and the relationship with other theories (chapter 2), according to the current state of the art of specific theories of rewriting. We have noticed that a TRS with a finite number of rules is not necessarily equivalent to other rewriting paradigms. We have checked expressiveness differences for ARSs, TRSs, STSs and PCSs. Moreover, a

more general and comprehensive study becomes interesting and necessary to treat them in more detail. This study involves expressiveness and complexity.

We have studied a motivating and simple calculus over names, $\lambda_\emptyset$, based only on the set of pure terms of classical λ-calculus (chapter 3). It can be seen as a good motivation for other calculi, and it results an interesting and pedagogical case study, including its de Bruijn versions.

What is a characterization? Roughly, a study or description of certain elements satisfying some property, for example being members of some set. In mathematics, a characterization is often understood as a "good description", i.e. a description of something which captures its main properties or structure. A manner of denoting some object which would not denote another object. We have studied perpetual reduction strategies in $\lambda\upsilon$ (chapter 4) from which we stated a characterization of the SN terms set. Such a characterization may entail inductive proofs for SN terms. Even when such a characterization exists, the set needs not to be decidable, as it happens with the SN terms set in all formulations of λ-calculus. But such an inductive proof proceeds anyway, allowing to "syntactically browse" all its terms for proving properties. A set could admit different characterizations. One may consider the number of characterizing elements (such as number of rules, their size), and perhaps the complexity of recognizing a member of the set. A relation with computational complexity issues is perhaps one of the better and more challenging aspects to pursue.

We have studied and established the weak confluence on open terms of a $\lambda\upsilon$-calculus with substitution interaction rules (chapter 5), which is also suitable to the more general CINNI. This makes a calculus in a different style than λs, but starting from a calculus having itself minimal rules, and leading to rule schemas. For these rule schemas critical pairs were generically analyzed.

We proved the weak normalization for simply typed λs_e on open terms (chapter 6), and gave a strategy for reaching their normal forms as well. We found another relevant difference between the styles, which is enough to deserve different treatment. For $\lambda\omega_e$ and $\lambda\omega'_e$, for example, the situation is slightly different that for the almost isomorphic λs_e due to the two-sorted character. $\lambda\omega'_e$ is a good example which shows that a calculus may not need more than a single index, when having adequate composition rules. Thus such a new calculus has a smaller set of terms when compared to its parent, but it is a subsystem as good as the whole system.

We have introduced a λ-calculus with constructors for modeling the pattern matching feature of functional programming (chapter 7). We have extensively studied commutation properties and confluence, with interest in not only the calculus but also in its subsystems, which justifies the divide-and-conquer method for assisting confluence proofs.

We have introduced the study of expansion in explicit substitution calculi (chapter 8), a point which had practically no attention in the literature. In a practical sense, the set of terms may have meaningless terms since when derivations starting from pure terms only matter. It

turns out that, for some de Bruijn calculi, if one ignores all terms which are not reachable from pure terms, then the set of terms results nonrecursive. We used explicit substitutions to code in a term the problem of deciding the existence of a common expansion between two terms, which was also proved to be undecidable. We think that the existence of a de Bruijn calculus (maybe as part of an abstract formulation) which has all the desirable properties, and where all terms which expand to pure terms form a recursive set (better yet, a context-free set), is a significant problem. Two de Bruijn formulations of the $\lambda_\emptyset$-calculus have the nice property that all the terms are pure, but these calculi are not PSN (chapter 3).

It is hard to imagine the future of rewriting formulations and theories. It is also easy to know that there is a future indeed, since there are many research tasks alive and others will emerge. We leave some problems open in this thesis too. Like substitution, we expect to formulate explicitly other aspects of rewriting, for the lessons learned showed that it is often interesting to make a change of paradigm every now and then.

We have stated lines of future work in the conclusion of each chapter. There are some research possibilities of special interest which we will mention below.

An interesting line of research is the study of other typing systems for the calculi treated in the previous chapters, such as intersection and higher-order typing.

When working with commuting diagrams within the divide-and-conquer method for proving confluence (chapter 7), it could be nice to formulate completeness results in relation to the "inferences" a program can perform. This is motivated by the fact that we have used just some specific lemmas which entail commutation results starting from axioms (as an initial table of hand-checked results), and perhaps other rules would do as well. Of special interest would be to apply this technique to explicit substitution calculi.

Another challenging task will be to analyze the expansion problem (chapter 8) for other calculi, in particular $\lambda\sigma$ and its variations. It could be interesting to find a relation with *Kolmogorov complexity*, that is to measure the "minimal" (pure, or other) terms which reduce in several steps to a given term.

We also believe that randomized rewriting formulations deserve attention - this means for example to relate randomness with the probability of a term being WN (or SN), as well as the probability of certain non-normalizing strategies to achieve a normal form when it exists.

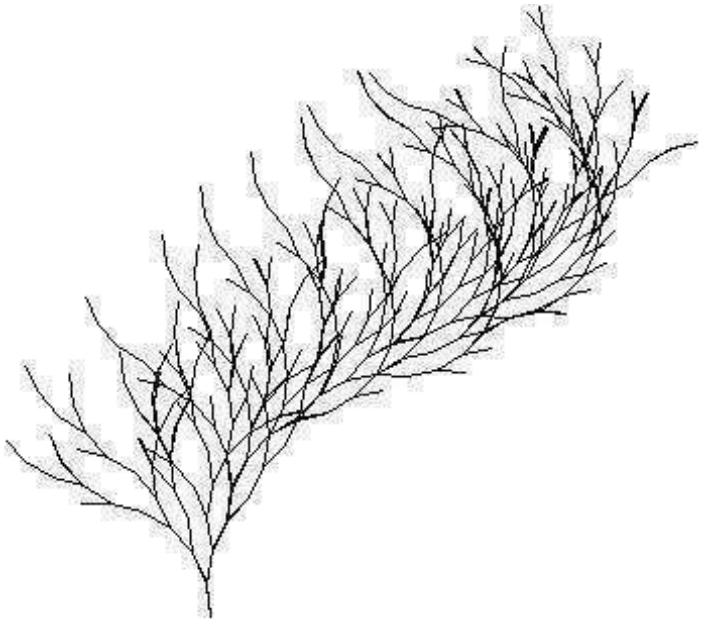


Figure 9.1: $(((S((S(SS))S))S)((((SS)S)(((SS)S)S))(SX)))$ after 60 left-most steps


Oye Manolo ... Esto de la reescritura ... que no me gusta tanto ...

References

[1] M. Abadi, L. Cardelli, P.-L. Curien and J.-J. Lévy. Explicit Substitutions. *Journal of Functional Programming*, 1(4):375–416, 1991. 7, 31

[2] H. Abelson and G. J. Sussman. Structure and Interpretation of Computer Programs. MIT Press, 1984. 7

[3] P. Aczel. An introduction to inductive definitions. In J. Barwise, editor, *Handbook of Mathematical Logic*, 739–782. North Holland, 1977. 78

[4] A. Arbiser, E. Bonelli and A. Ríos. Perpetuality in a Lambda Calculus with Explicit Substitution and Composition. *Proc. WAIT 2002, 31 JAIIO*, Santa Fe, 2002. 4, 98, 102, 254

[5] A. Arbiser. On the expressive power of calculi with explicit substitution. *UNILOG 2005*, Montreux, 2005. 105

[6] A. Arbiser. Abstract reduction systems and subsystems: understanding sub-calculi and strategies. *9th. Asian Logic Conference 2005*, Novosibirsk, 2005.

[7] A. Arbiser, F. Kamareddine and A. Ríos. The Weak Normalization of the Simply-Typed λs_e. J. of the IGPL. To appear (2005). 134

[8] A. Arbiser, A. Miquel and A. Ríos. A Lambda Calculus with Constructors. *Proc. RTA '2006* (to appear). 165, 206

[9] F. Baader and T. Nipkow. Rewriting and All That. Cambridge Univ Press, 1999. 12, 13, 14, 15, 42, 45, 62, 77

[10] H. P. Barendregt, J. A. Bergstra, J. W. Klop and H. Volken. Degress, reductions and representability in the lambda calculus. Technical Report 22, Utrecht University, 1976.

[11] H. P. Barendregt. *The Lambda Calculus: its Syntax and Semantics*. Studies in Logic and the Foundations of Mathematics 103. North-Holland, Amsterdam, revised edition, 1984. 4, 7, 16, 17, 18, 21, 32, 40, 53, 70, 78, 164, 165, 214, 279, 280

[12] H. P. Barendregt. *Lambda Calculi with Types*. In: Handbook of Logic in Computer Science. North-Holland, 1999. 4, 32, 68, 70, 214, 233

[13] Z. Benaissa, D. Briaud, P. Lescanne and J. Rouyer-Degli. $\lambda\upsilon$, a calculus of explicit substitutions which preserves strong normalisation. *Functional Programming*, 6(5), 1996. 7, 23, 25, 28, 77, 79, 80, 81, 82, 85, 106, 111, 130, 235

[14] M. Bezem, J.W. Klop and R. de Vrijer (eds.) *Term Rewriting Systems (TeReSe)*. Cambridge University Press, 2003. 4, 12, 13, 14, 15, 39, 44, 47, 55, 62, 77, 279

[15] R. Bloo. *Preservation of Termination for Explicit Substitutions*. PhD thesis, Eindhoven University, 1997. 7, 22, 23, 27, 32, 56, 77, 105

[16] C. Böhm, M. Dezani-Ciancaglini, P. Peretti, and S. Ronchi Della Rocha. A discrimination algorithm inside lambda-beta-calculus. *Theoretical Computer Science*, 8(3):265–291, 1979. 164, 165

[17] E. Bonelli. Perpetuality in a Named Lambda Calculus with Explicit Substitutions. *MSCS*, 11(1), 2001. 78, 79, 81, 85, 89, 98

[18] S. Cerrito and D. Kesner. Pattern matching as cut elimination. *Logics In Computer Science (LICS'99)*, 98–108, 1999. 6, 164

[19] A. Church and J.B. Rosser. Some properties of conversion. *Transactions of the American Mathematical Society*, 39, 1936. 4, 10, 18, 164

[20] A. Church. *The calculi of lambda-conversion*, volume 6 of *Annals of Mathematical Studies*. Princeton, 1941. 4, 10, 18

[21] H. Cirstea and C. Kirchner. Rho-calculus, the rewriting calculus. *5th International Workshop on Constraints in Computational Logics*, 1998. 6, 164

[22] H. Cirstea and C. Kirchner. The rewriting calculus - Part I. L. J. of the IGPL, 9(3), 2001. 6

[23] E. Contejean and C. Marché. CiME: Completion Modulo E. In H. Ganzinger, ed. *Proc. RTA '96*, LNCS 1103, 416–419. Springer Verlag, 1996. Available at `http://cime.lri.fr`. 122

[24] P.-L. Curien. Categorical Combinators, Sequential Algorithms and Functional Programming. Pitman, 1986. Revised edition, Birkhäuser (1993). 7, 31

[25] P.-L. Curien, T. Hardin and J.-J. Lévy. Confluence properties of weak and strong calculi of explicit substitutions. Technical Report RR 1617, INRIA, Rocquencourt, 1992. 31

[26] R. David and B. Guillaume. A λ-calculus with explicit weakening and explicit substitutions. *MSCS*, 11(1), 2001. 102

[27] N. de Bruijn. Lambda-Calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser Theorem. *Indag. Mat.*, 34(5):381–392, 1972. 5, 19, 51

[28] N. Dershowitz. Termination of Rewriting. Journal of Symbolic Computation, 3(1 and 2), 69–116, 1987.

[29] R. Di Cosmo, D. Kesner and E. Polonovsky. Proof nets and explicit substitutions. *FOSSACS'2000*, LNCS 1784, Springer-Verlag, 2000.

[30] D. Dougherty and P. Lescanne. Reductions, intersection types and explicit substitutions. *Proc. of TLCA*, LNCS 2044, Springer-Verlag, 2001. 23

[31] M. Ferreira, D. Kesner and L. Puel. Lambda-calculi with explicit substitutions and composition which preserve beta-strong normalization. Proc. 5th International Conference on Algebraic and Logic Programming, Aachen, Germany, September 1996. LNCS 1139: 284–298, Springer-Verlag, 1996. 106

[32] M. Ferreira, D. Kesner and L. Puel. Lambda-calculi with Explicit Substitutions and Weak Composition Preserving Strong Normalization. Applicable Algebra in Engineering, Communication and Computing 9(4), 333–371, 1999. 106

[33] J. H. Gallier. On Girard's "candidats de reducibilité". In P. Odifreddi, editor, *Logic in Computer Science*, number 31 in APIC, 123–203. Academic Press, 1990.

[34] J.-Y. Girard. Locus solum: From the rules of logic to the logic of rules. *Mathematical Structures in Computer Science*, 11(3):301–506, 2001. 165

[35] B. Guillaume. *Un calcul des substitutions avec étiquettes.* PhD thesis, Université de Savoie, Chambéry, 1999. 77, 102

[36] B. Gramlich. *Termination and Confluence Properties of Structured Rewrite Systems.* PhD thesis, Fachbereich Informatik der Universität Kaiserslautern, 1996.

[37] J. Goubault-Larrecq. A proof of weak termination of the simply typed $\lambda\sigma$-calculus. Proc. Int. Workshop on Types for Proofs and Programs (TYPES'96). LNCS 1512, 134–151, Springer-Verlag, 1998. 134, 144, 151, 160

[38] Th. Hardin and J.-J. Levy. A confluent calculus of substitutions. France-Japan Artificial Intelligence and Computer Science Symposium, 1989.

[39] J. Hindley and J. Seldin, editors. To H.B. Curry: Essays on Combinatory Logic. Academic Press, 1980. 279

[40] P. Hudak, S. Peyton-Jones, and P. Wadler. Report on the programming language Haskell, a non-strict, purely functional language (Version 1.2). Sigplan Notices, 1992. 6, 164

[41] C. Barry Jay. The pattern calculus. *ACM Transactions on Programming Languages and Systems*, 26(6):911–937, 2004. 6, 164

[42] W. Kahl. Basic pattern matching calculi: Syntax, reduction, confluence, and normalisation. Technical Report 16, Software Quality Research Laboratory, McMaster Univ., 2003. 6, 164

[43] F. Kamareddine and A. Ríos. A λ-calculus à la de Bruijn with explicit substitutions. Proceedings of PLILP'95. LNCS 982, 45–62, Springer-Verlag, 1995. 20, 25, 27, 60, 77

[44] F. Kamareddine and A. Ríos. Bridging de Bruijn indices and variable names in explicit substitution calculi. Technical Report, University of Glasgow, 1996. 20

[45] F. Kamareddine and A. Ríos. Extending a λ-calculus with explicit substitution which preserves strong normalisation into a confluent calculus on open terms. Technical Report, University of Glasgow, 1996. 27, 106

[46] F. Kamareddine and A. Ríos. Extending a λ-calculus with explicit substitution which preserves strong normalisation into a confluent calculus on open terms. *Journal of Functional Programming*, 7(4):395–420, 1997. 7, 27, 30, 106, 134, 142, 143, 155

[47] F. Kamareddine and A. Ríos. Relating the $\lambda\sigma$- and λs-Styles of Explicit Substitutions. *Journal of Logic and Computation*, 10(3):349–380, 2000. 7, 21, 28, 30, 105, 133, 134, 138, 142, 250, 251

[48] D. Kesner. Confluence of Extensional and Non-Extensional lambda-calculi with Explicit Substitutions. TCS, 238 1-2, 183–220, 2000. 106

[49] Z. Khasidashvili. The Longest Perpetual Reductions in Orthogonal Expression Reduction Systems. *Proc. of the 3rd. International Conference LFCS*, LNCS 813, Springer-Verlag, 1994. 77

[50] Z. Khasidashvili. On Longest Perpetual Reductions in Orthogonal Expression Reduction Systems. *TCS*, 2001. 77

[51] Z. Khasidashvili, M. Ogawa and V. van Oostrom. Perpetuality and uniform normalization in orthogonal rewrite systems. *Information and Computation*, 164:118–151, 2001. 77

[52] Z. Khasidashvili, M. Ogawa and V. van Oostrom. Uniform Normalization Beyond Orthogonality. *Proc. RTA 2001*, LNCS 2051, Springer-Verlag, 2001. 77

[53] C. Kirchner, H. Kirchner, and M.Vittek. Designing CLP using Computational Systems. In P. Van Hentenryck and S. Saraswat, eds., *Principles and Practice of Constraint Programming*. The MIT press, 1995. 122

[54] J. W. Klop. *Combinatory Reduction Systems*. PhD thesis, Mathematical Centre Tracts n.127, CWI, 1980. 8

[55] D. E. Knuth and P.B. Bendix. Simple Word Problems in Universal Algebras. In J. Leech, editor, Computational Problems in Abstract Algebra, 263–297. Pergamon Press, Oxford, 1970.

[56] K. Kuratowski. *General Topology*. Academic Press, 1966. 38

[57] D. S. Lankford. On Proving Term Rewriting Systems are Noetherian. Technical Report, Louisiana Tech. University, Mathematics Dept., Ruston LA, 1979.

[58] P. Lescanne. Implementation of Completion by Transition Rules + Control: ORME. In H. Kirchner and W. Wechler, editors, Proc. 2nd Int. Conference on Algebraic and Logic Programming, Nancy (France), LNCS 463, 262–269, Springer-Verlag, 1990.

[59] P. Lescanne. *From Lambda-sigma to Lambda-upsilon: a Journey Through Calculi of Explicit Substitutions*. Proc. POPL 1994. 7

[60] P. Lescanne and J. Rouyer-Degli. The Calculus of Explicit Substitutions $\lambda\upsilon$. CNRS and INRIA-Lorraine, 1995. 23, 25, 106, 107, 111, 130, 234, 236, 248

[61] S. Lucas. Reescritura con Restricciones de Reemplazamiento. PhD thesis, Univ. Pol. Valencia, 1998. 40

[62] S. Lucas. Context-sensitive rewriting, lazy rewriting, and on-demand rewriting. DSIC, Univ. Pol. Valencia, 2002. 40

[63] L. Maranget. Optimal derivations in weak lambda-calculi and in orthogonal TRS. *Proc. of POPL*, 1991. 23, 32

[64] U. Martin and T. Nipkow. Ordered Rewriting and Confluence. In M. E. Stickel, editor, Proc. 10th Int. Conference on Automated Deduction, Kaiserslautern (Germany), LNCS 449, 366–380, Springer-Verlag, 1990.

[65] N. Martì-Oliet and J. Meseguer. Rewriting Logic as a Logical and Semantical Framework. TR SRI-CSL-93-05, August 1993. 6

[66] P.-A. Melliès. *Description Abstraite des Systèmes de Réécriture*. PhD thesis, Université Paris VII, 1996. 31, 32, 51, 58, 105

[67] P.-A. Melliès. Axiomatic Rewriting Theory II: The $\lambda\sigma$-calculus enjoys finite normalisation cones. *Journal of Logic and Computation*, 10(3):461–487, 2000. 23

[68] R. Milner, M. Tofte, and R. Harper. *The definition of Standard ML*. MIT Press, 1990. 6, 164

[69] C. Muñoz. Un calcul de substitutions pour la représentation de preuves partielles en théorie de types. PhD thesis, Université Paris VII. 1997.

[70] The Objective Caml language. `http://caml.inria.fr/`. 6, 164

[71] M. J. O'Donnell. Term-Rewriting Implementation of Equational Logic Programming *Proc. RTA '87*, LNCS 256, Springer-Verlag, 1987. 6

[72] E. Polonovski. Substitutions explicites, logique et normalisation. PhD thesis, Université de Paris 7, 2004. 222

[73] G. Revesz. Axioms for the Theory of Lambda-Conversion. SIAM 14(2), SICOMP, 1985. 51

[74] A. Ríos and F. Kamareddine. The λs-calculus: its typed and its extended versions. Technical report, Dept. of Computing Science, University of Glasgow, 1995. 27

[75] A. Ríos. *Contribution à l'étude des λ-calculs avec substitutions explicites*. PhD thesis, Université de Paris 7, 1993. 79, 105, 177

[76] K.H. Rose. Explicit cyclic substitutions. *Proc. of CTRS*, LNCS 656, Springer-Verlag, 1992. 22

[77] K.H. Rose. Explicit Substitution - Tutorial & Survey. University of Aarhus, 1996. `http://www.brics.dk/LS/96/3/BRICS-LS-96-3/BRICS-LS-96-3.html`. 51, 106

[78] G. Rozenberg et al. Handbook of Graph-Grammars and Computing by Graph Transformation. Vol I. World Scientific, 1997. 49

[79] M.-O. Stehr. Programming, Specification and Interactive Theorem Proving. Towards a Unified Language based on Equational Logic, Rewriting Logic and Type Theory. PhD thesis, Universitat Hamburg, 2002. 8, 106, 112, 125

[80] V. van Oostrom. Lambda calculus with patterns. Technical Report IR-228, Vrije Universiteit, Amsterdam, 1990. 6, 164

[81] F. van Raamsdonk. Higher-Order Rewriting *Proc. RTA '99*, LNCS 256, Springer-Verlag, 1999. 55

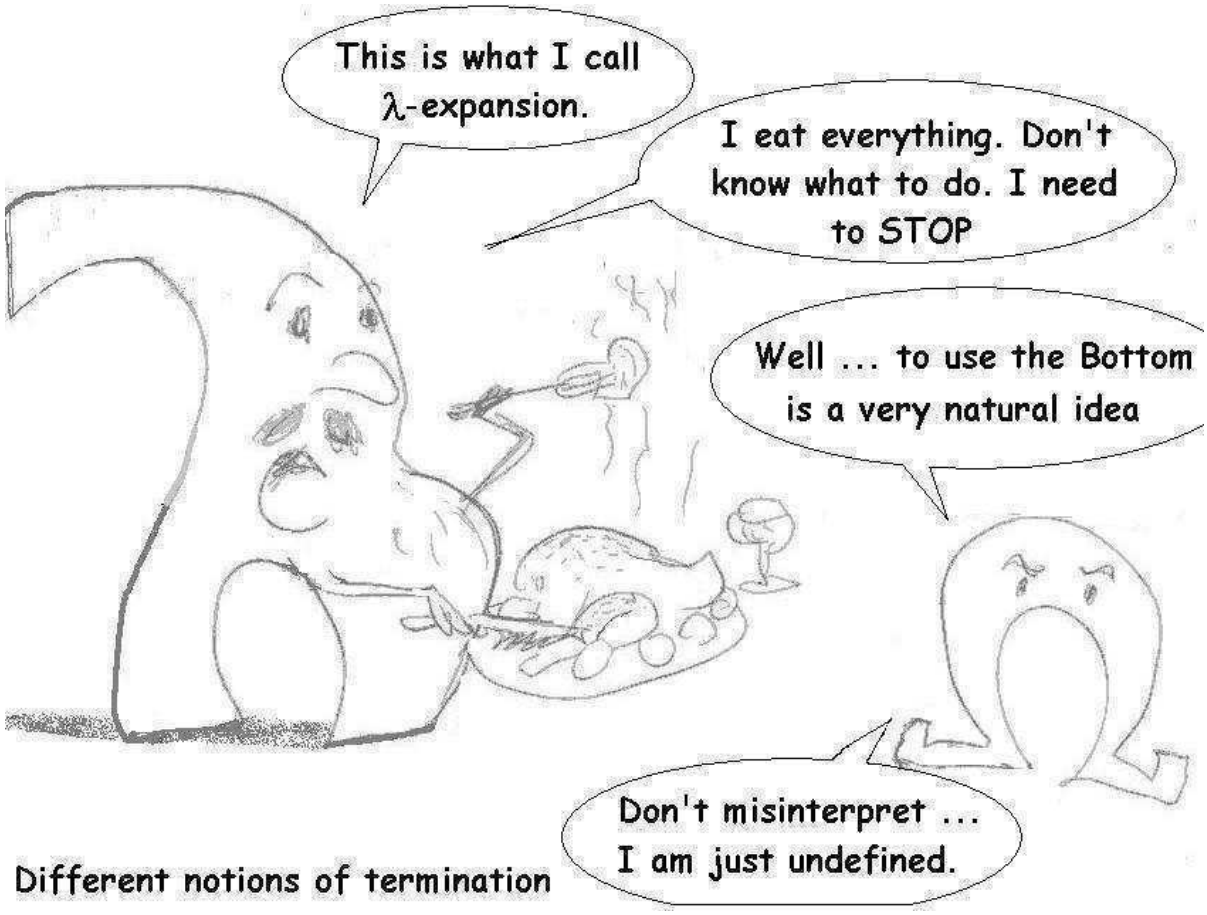
[82] F. van Raamsdonk, P. Severi, M.H. Sørensen and H. Xi. Perpetual Reductions in Lambda Calculus. *Journal of Information and Computation*, 149(2), 1999. 77, 78, 89, 98, 102, 254

[83] F. van Raamsdonk. Confluence and Normalization for Higher-Order Rewriting. PhD thesis, Amsterdam University, 1996. 8

[84] J. Waldmann. The Combinator S. PhD thesis, Friedrich-Schiller-Universität Jena, 1998. 49, 222

[85] H. Zantema. Termination of Term Rewriting: Interpretion and Type Elimination. Journal of Symbolic Computation, 17(1), 23–50, January 1994.

[86] H. Zantema and A. Geser. Non-looping Rewriting. TR UU-CS1996-03, Utrech University, 1996.



Different notions of termination

Appendix A

Implementation

That all our knowledge begins with experience there should be no doubt. – I. Kant

ABSTRACT We give in this appendix a brief description of the related software tools provided with this thesis or used in relation to some of the topics covered in the previous chapters. We do not provide detailed descriptions nor instructions, just a minimal account about the goals and functionality of these experimental tools. Their purpose is only to illustrate some rewriting concepts in a practical way and to allow experimentation with them, as well as to see by oneself how some terms, types and derivations may look like.

A.1 Introduction

We are also interested in the implementation of various rewriting systems. Even when there is still no clear or complete concept of implementation, and that this could be attacked from different points of view, we remark that explicit substitutions constitute a way of implementing lambda calculus.

Now, since calculi of explicit substitution with de Bruijn indices are TRSs, we implemented a code generator which (more generally) transforms a given Context-Sensitive Rewriting System to a Haskell program implementing some specific strategy (so each function symbol can be evaluated in an outermost or innermost way, and also the contextual rewriting can be restricted to a given subset of its arguments). The idea of the implementation is to find (and experiment with) appropriate strategies for (possibly context-sensitive) lambda calculi with explicit substitution, where the strategy may depend on each function symbol.

This chapter just summarizes different tools implemented for helping us in part of our research.

A.2 Commuter

`Commuter` is a tool which allows to specify commutation pairs of subsystems of a rewriting system. Chapter 7 includes a "real case analysis".

A.2.1 Working with `commuter`

The user can mark commutation pairs (for instance by inputting them from a commutation specification file given in Ascii format), use the inference rules arbitrarily and look how the entire commutation grid looks like at any time. The system generates minimal proofs for weak commutation and commutation of pairs of subsystems (and weak confluence and confluence of the subsystems).

Alternatively, the system generates a minimal set of proof trees for the weak confluence (if valid), for all the systems, using an algorithm which is linear in the number of subsystems of the main system (see the Appendix in chapter 7, for concrete examples).

A.2.2 Handling of abstract closure conditions

`Commuter` can work with any finite set of (binary) closure conditions.

The proofs output could be "short" in a specific way. The program uses a criterion on how to minimize the redundancies of a set of proofs for a given system and subsystems. We call redundant a sub-tree which repeats itself along the set of trees, i.e. a tree may be a proper sub-tree of a bigger one. For such a closure condition set, we generate the proofs semi-automatically, in such a way that the number of redundancies is minimized.

The output commutation grid for λx-calculus appears in Figure A.1. Since λx has 5 rules, this makes $2^5 = 32$ subsystems. Each pixel represents a pair of subsystems, starting from the upper-left (both empty subsystems) to the bottom-right (both full systems), and its color indicates confluence (grey), weak confluence (black) or neither (white).

The output commutation grid for the $\lambda\mathcal{B}_{\mathcal{C}}$-calculus appears in Figure 7.6 (chapter 7).

More information and examples of `Commuter` can be found at the URL
http://www.dc.uba.ar/people/materias/reescritura/commuter

A.3 From rewriting to a functional program

`trs2code` is a code generator which transforms a given context-sensitive rewriting system to a Haskell program implementing some of a given number of specific strategies (so each function

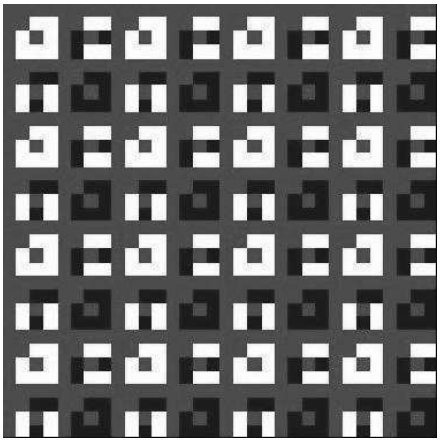


Figure A.1: λx-calculus weak commutation grid

symbol can be evaluated in an outermost or innermost way, and also the contextual rewriting can be restricted to a given subset of its arguments). The strategy may depend on each function symbol.

As a simple but significative example, the $\lambda\upsilon$-calculus, considering open terms, can be identified to a sub-ARS of the one-sorted TRS with the following rules:

$$
\begin{gathered}
ap(la(x), y) \to cl(x, sl(y)) \\
cl(ap(x, y), s) \to ap(cl(x, s), cl(y, s)) \\
cl(la(x), s) \to la(cl(x, li(s))) \\
cl(one, sl(x)) \to x \\
cl(suc(n), sl(x)) \to n \\
cl(one, li(s)) \to one \\
cl(suc(n), li(s)) \to cl(cl(n, s), sh) \\
cl(n, sh) \to suc(n)
\end{gathered}
$$

where x, y, s, n are variables. These rules serve as a specification for obtaining Haskell code implementing leftmost-outermost, leftmost-innermost, and other reduction strategies, given by the usual two-sorted (terms and substitutions) algebra or as a sub-calculus of a one-sorted algebra. As an example, the following code is obtained for a leftmost-outermost strategy:

```
-- type definition

data Term = N Nu | Cl Term Subs | Ap Term Term | La Term
data Nu = One | Suc Nu
data Subs = Sl Term | Sh | Li Subs

-- syntactic equality functions
```

```
eqn One One = True
eqn (Suc x) (Suc y) = eqn x y
eqn x y = False

eq (N x) (N y) = eqn x y
eq (Cl x s) (Cl y t) = (eq x y) && (eqs s t)
eq (Ap x1 x2) (Ap y1 y2) = (eq x1 y1) && (eq x2 y2)
eq (La x1) (La y1) = (eq x1 y1)
eq x y = False

eqs (Sl x1) (Sl y1) = (eq x1 y1)
eqs Sh Sh = True
eqs (Li s) (Li t) = eqs s t
eqs s t = False

neq x y = not (eq x y)
neqs s t = not (eqs s t)

-- calculus rules

--terms
r (Ap (La x) y) = (Cl x (Sl y))
r (Ap z1 z2) =
 let rz1 = r z1 in
  if (neq rz1 z1) then
  (Ap rz1 z2)
  else
  let rz2 = r z2 in
   if (neq rz2 z2) then
   (Ap z1 rz2)
   else
   (Ap z1 z2)

r (Cl (Ap x y) s) = Ap (Cl x s) (Cl y s)
r (Cl (La x) s) = La (Cl x s)
r (Cl (N one) (Sl x)) = x
r (Cl (N (Suc n)) (Sl x)) = N n
r (Cl (N One) (Li s)) = N One
r (Cl (N (Suc n)) (Li s)) = Cl (Cl (N n) s) Sh
```

```
r (Cl (N n) Sh) = N (Suc n)
r (Cl z1 z2) =
 let rz1 = r z1 in
  if (neq rz1 z1) then
  (Cl rz1 z2)
  else
  let rz2 = rs z2 in
   if (neqs rz2 z2) then
   (Cl z1 rz2)
   else
   (Cl z1 z2)
r z = z

--substitutions
rs (Li s) = Li (rs s)
rs (Sl x) = Sl (r x)
rs s = s

--rewrite a given number of steps
rn(0,z) = z
rn(n,z) = r(rn(n-1,z))

--rewrite to normal form if it exists
rnf(z) =
 let rz = r z in
 if (eq rz z) then z
 else rnf(rz)
```

As a practical application, which could be useful for testing rewriting scenarios, we implemented in Haskell some rewriting systems for experimentation: classical λ-calculus, S, K, I-combinatory logic (CL) (11; 14; 39), $\lambda s/se$, $\lambda\upsilon$ and λw_s, with data types for the sets of terms and substitutions, some strategies and pretty-printing routines. This implementation is an extension of the Haskell code produced by `trs2code`. In the case of $\lambda\upsilon$ the code also provides a way of handling generic closures (i.e. representing "explicitly" the level of iterated lifts, which is common notation in previous chapters) as well as meta-variables. More information and examples of `trs2code` and related programs can be found at the URL
http://www.dc.uba.ar/people/materias/reescritura/trs2code.

A.3.1 Implementing types

Simply typed $\lambda\upsilon$ and other calculi were implemented as well. Given a term or a substitution, and a typing context of zero or more types, the system infers its type in that context or if it has no type.

Every typing rule has been implemented as a defining clause for a function. A bottom element is returned when the input term or substitution is not typable. This method of implementing type inference may lead to an extension to handle polymorphic versions of some calculi.

A.3.2 CL to λ-calculus

We implemented also the conversion from λ-calculus to CL (and viceversa), based on the classical mechanism (11). The interesting point is that the conversion is done by rewriting rules. The language syntax combines the one of classical λ-calculus with constant terms. The rewriting system for conversion from λ-calculus to CL is formulated below.

The syntax is as follows:

$$M \quad ::= \quad x \mid \lambda x.M \mid MM \mid S \mid K$$

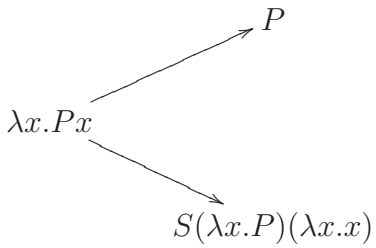
where S and K are constants. The rules are:

$$\begin{array}{lcll} \lambda x.x & \to & SKK & \\ \lambda x.M & \to & KM & x \notin FV(M) \\ \lambda x.PQ & \to & S(\lambda x.P)(\lambda x.Q) & x \in FV(PQ) \\ \lambda x.Px & \to & P & x \notin FV(P) \end{array}$$

Do not confuse this system, for performing transformations, with the actual CL-rewriting system or with λ-calculus.

We use leftmost/parallel reduction, and a conversion function rewrites a given term (starting mixed λ-calculus and CL) until a normal form is reached (the CL version).

Although the last rule (η-reduction) is optional, its purpose is to avoid very long conversions, both in size and in number of steps. Without that rule, this rewriting system is canonical, but adding this rule generates the following critical pair:



Thus it is still SN but not confluent. The pair can be closed by adding the rule

$$S(Kx)(SKK) \to x$$

We did not include actual CL-rewrite steps, such as $Kxy \to x$, in this conversion system. In that way more efficient translations could be obtained. This is left for future analysis.

A.4 Fractal objects

Every CL-combinator can be pictured as a tree, where the application operation defines the sub-tree relation, and constants and variables are interpreted as leaves. Between chapters there are images illustrating samples of CL-terms after many-step leftmost derivations.

`frint` is a tool which can produce this graphical output from arbitrary TRSs (for example, CL), as well as for experimenting with L-systems, a "parallel" variant of the STSs in which each symbol has a "graphical meaning". The idea is taken from the FracTree 1.0 software by M. Schernau (1993) and the book from Pleitgen and Saupe (eds.), *The Science of Fractal Images*, Springer-Verlag, NY 1988.

The formulation we handle is built on strings, where one starts with an initial expression (axiom) and then rewriting rules are applied (in parallel) over that expression, a certain number of times. After the iteration, a graphical interpretation is done. This means that the symbols of the resulting string are "drawn" in different ways.

At every discrete moment a current *direction* is defined (an angle ranging from 0 to 2π) as well as a current *displacement* (a positive real number). The rules should contain characters whose graphical meanings are described as follows:

- F : moves the pen forward to the current direction, drawing a segment, using current displacement value
- f : moves the pen forward to the current direction but without drawing, using current displacement value
- $+$,$-$: rotates the pen heading $\frac{2\pi}{direction}$ right or left respectively
- $|$: rotates the pen heading π
- $*$: multiplies by 2 the displacement to use
- $/$: divides into half the displacement to use
- $[\ldots]$: executes the given block and then restores the pen position

Input data are:

- *dir* : number of directions (different angles), indicating how the circumference is divided for rotations

- Ax : axiom, i.e. starting expression
- Rules : set of rewriting rules

The informal algorithm is the following:

```
Procedure to draw the n-th generation from an axiom E
 repeat for i = 1 to n
   for each rule L -> R
      for each subexpression S in E matching with L
         compute S' and R' given the matching substitution
         change in E all S' by R'
 draw E
end
```

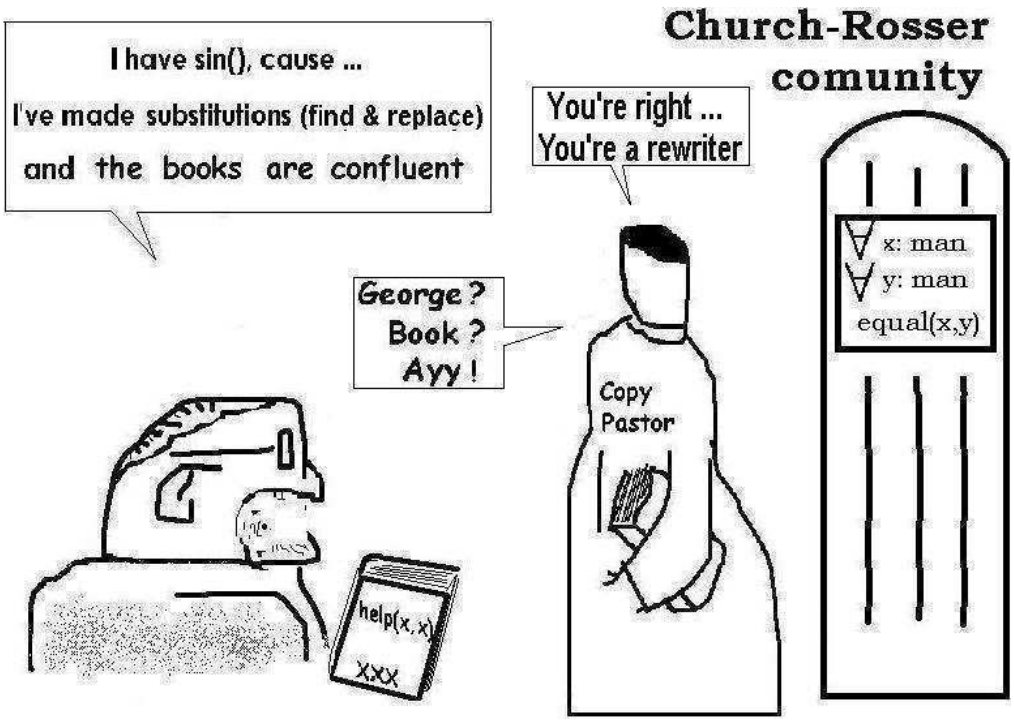
More information and examples of `frint` can be found at the URL
http://www.dc.uba.ar/people/materias/reescritura/frint.

A.5 Term generation

Given a calculus, term generation is useful for testing properties or conjectures over families of terms empirically before looking for a formal proof.

`Termgen` is a tool that generates terms of several λ-calculus variants: λ-calculus a la de Bruijn, $\lambda\upsilon$, λs and $\lambda\sigma$. After specifying the maximum depth and size, the terms are generated in Haskell, LaTeX or Ascii format.

More information and examples of `termgen` can be found at the URL
http://www.dc.uba.ar/people/materias/reescritura/termgen.


Church-Rosser
comunity
I have sin(), cause ...
I've made substitutions (find & replace)
and the books are confluent
You're right ...
You're a rewriter
George ?
Book ?
Ayy !
Copy
Pastor
help(x, x)
XXX
x: man
y: man
equal(x,y)

Appendix B

Glossary

This chapter consists of an informal glossary which can be consulted for quick reference during a first reading of the thesis. It includes most of the concepts treated in the preceding chapters, and their definitions are given in an intuitive and informal manner. For formal definitions of these and other concepts, see the thesis preliminaries chapter as well as the introductory paragraphs and the definitions given in each single chapter.

- abstract rewriting system (ARS): a set and a binary relation over it
- alpha-conversion ($=_\alpha$): the possibility of renaming some bound variables in a term preserving its semantics
- algorithm: a mechanical procedure which can solve a problem or perform a given task; an abstraction of a program to implement
- beta-reduction ($\rightarrow_\beta$): a reduction rule which computes the simultaneous substitution of a given term for a given variable in a given term
- beta-rule: a rule in a calculus of explicit substitution which plays the role of the β-reduction, usually creating a closure which will need to propagate within the term
- beta-equivalence ($=_\beta$): the least equivalence relation which includes the relation $\rightarrow_\beta$; somehow reflecting that two given terms have the same computational meaning
- beta-eta-equivalence ($=_{\beta\eta}$): the least equivalence relation which includes the relation $\rightarrow_{\beta\eta}$; somehow reflecting that two given terms have the same computational meaning when considering extensionality
- calculus: a rewriting system based on rules

- calculus of explicit substitution: a rewriting system which, allows the computational power of λ-calculus by including the substitution object in the term syntax, and which propagate substitutions (once these appear in terms by means of the corresponding beta-rule)
- canonicity: normality, something expected, typical or standard; also, a rewriting system is canonical when it is CR and SN
- characterization: a manner of describing a set of objects which allow to (easily) identify them; a description of properties which a given set of objects satisfy and no other does
- Church-Rosser (CR): the property of confluence
- closure: a substitution represented in syntax; also a term with such a substitution pending
- combinator: a term without free variables
- combinatory logic: a rewriting system which handles combinators from lambda-calculus, also presented as a TRS with two constants (S and K), an application and two rewriting rules
- commutation: the possibility of, from a single term, joining any pair of divergent reductions, by applying reduction rules which occur at the opposite sides in the diagram
- commutation, strong: commutation where one side of the diagram is closed with 0 or 1 step
- commutation, weak: commutation where the diagram starts with two divergent single-steps instead of arbitrary length
- complexity: a branch of Computer Science studying the computational difficulty of solving different problems, according to the space or time that the solution may require
- composition rule: a rule which allows two substitutions (closures) to interact, possibly combining them into another substitution (closure)
- computable: same as recursive or decidable
- computability: a branch of Computer Science studying and describing the problems which can or cannot be solved by an algorithm, as well as the characteristics which different computing formulations and paradigms may turn a problem into solvable
- confluence: the property which ensures that every divergent diagram can be closed
- confluence, strong: confluence where one side of the diagram is a derivation of 0 or 1 step

- confluence, weak: confluence where the diagram starts with two divergent single steps
- constant: in first order signatures, a function symbol of arity 0
- constructor: a constant used to represent structured data when using pattern matching
- context: same as environment, in typed λ-calculi, a sequence of types or a set of pairs of the form variable-type
- context-sensitive rewriting system (CSRS): a TRS where rewriting is allowed or restricted under specific positions/parameters of function symbols
- de Bruijn index: a natural number representing a variable in a λ-term, which denotes the number of λ-binders over it until the binder it abstracts to
- decidable: the property of existence of an algorithm to solve a given problem
- derivation: a reduction of 0 or more steps, a sequence of reductions, i.e. rewriting steps from a given starting object
- diamond property: the property which establishes that any divergence diagram with a single step on each side can be closed by another step on each side
- divergence: a reduction diagram where two derivations begin with the same term
- environment: see context
- eta-reduction ($=_\eta$): a reduction rule which computes from the term $\lambda x.Mx$ the term M whenever x does not occur free in M
- equivalence: a relation which is reflexive, symmetrical and transitive
- expansion: the reverse of reduction; a term expands to another one if and only if the latter reduces into the former
- extensionality: the property and axiom which enables to interpret a term like $\lambda x.Mx$ as M, see η-reduction; it corresponds to the fact that two functions should be identified when they yield the same result when applied to the same values
- function: a relation which for every element (in the domain) there is one and only one element related (in the codomain)
- function, computable: a function for which an algorithm exists to calculating it for every element in its domain

- grammar: a formal system having as goal the description of a language, usually given by a set of non-terminal symbols, a set of terminal symbols, a set of rules (pairs of strings) and a starting symbol; a given string of terminal symbols is in the language generated by the grammar if and only if there exists a derivation from the starting symbol to the given string
- grammar, context-free: a grammar where the left-hand side of each rule consists of a single non-terminal symbol
- grammar, context-sensitive: a grammar where the left-hand side of each rule consists of any string of symbols of size less or equal than the corresponding right-hand side
- graph, directed: a collection of nodes (arbitrary elements of a set) and edges (ordered pairs of nodes)
- graph, undirected: a collection of nodes (arbitrary elements of a set) and edges (unordered pairs of nodes)
- interpretation method: a technique for proving confluence by relating different reduction relations, namely by identifying a relation over the normal forms of another relation
- label: a mark or tag which may be applied to sub-terms of a given term, in some specific term set, sometimes in order to follow-up or trace reduction steps
- labeling: a technique which allows to assign labels (marks) to some sub-terms of specific terms
- lambda-calculus (λ): a rewriting system, introduced by A. Church in the 1930's with the goal of describing most mathematical theories, in particular computing; it includes the use of variables, applications and abstractions, and serves as a basis to the functional programming paradigm, and the description of (some) logical theories
- lambda-omega ($\lambda\omega$): a calculus with explicit substitution over de Bruijn indices
- lambda-omega-e ($\lambda\omega_e$): a calculus with explicit substitution over de Bruijn indices and composition rules
- lambda-s (λs): a calculus with explicit substitution over de Bruijn indices
- lambda-se (λs_e): a calculus with explicit substitution over de Bruijn indices and composition rules confluent on open terms
- lambda-sigma ($\lambda\sigma$): a two-sorted calculus with explicit substitution over de Bruijn indices and a big number of rules

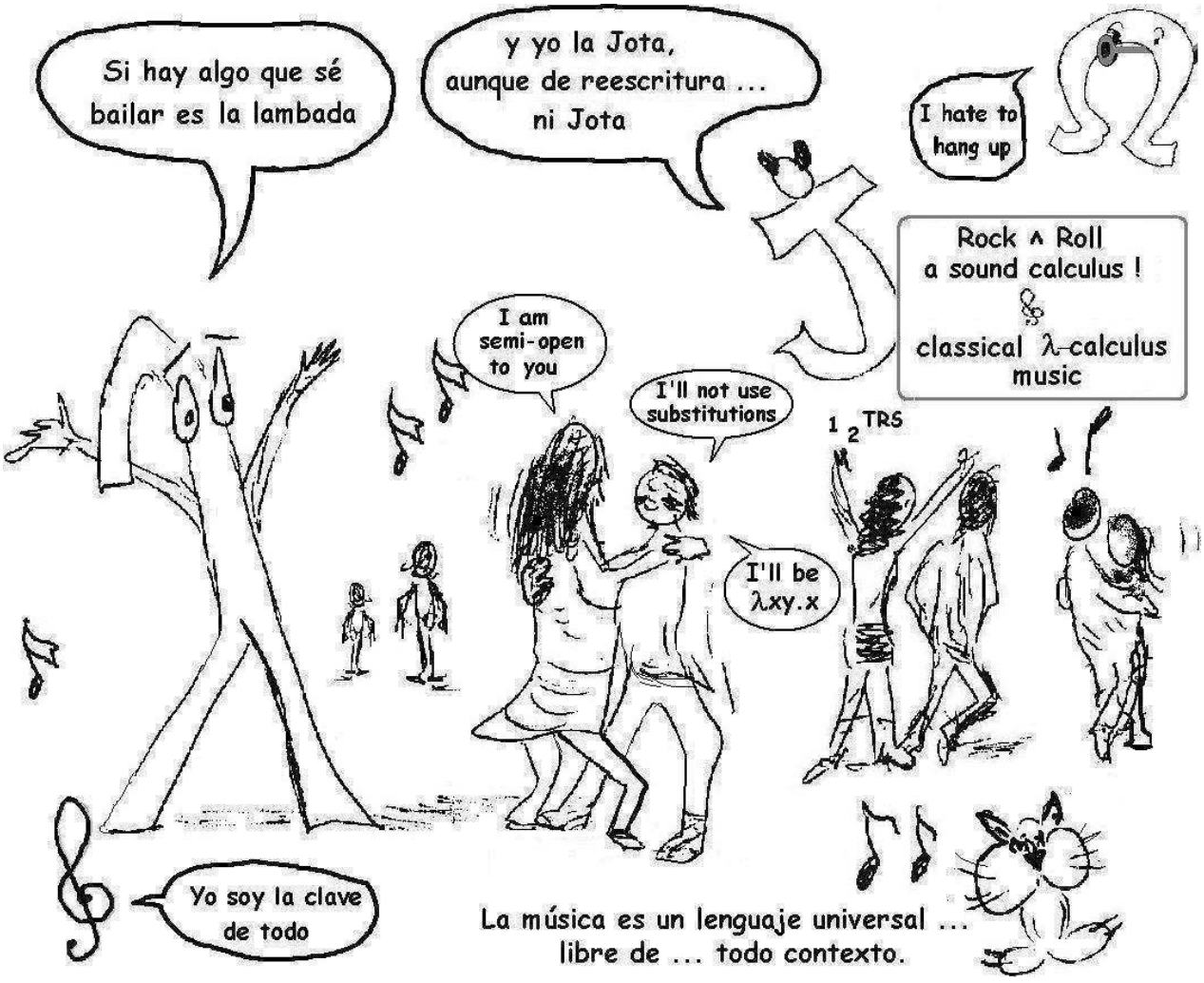
- lambda-upsilon ($\lambda\upsilon$): a two-sorted calculus with explicit substitution over de Bruijn indices
- lambda-w (λw): a calculus with weakening over de Bruijn indices
- lambda-ws (λw_s): a calculus with explicit substitution and weakening over de Bruijn indices
- lambda-x ($\lambda\mathtt{x}$): a calculus with explicit substitution with names
- lambda-xc ($\lambda\mathtt{x}_c$): a calculus with explicit substitution with names and a composition rule
- language: a set of strings from a given alphabet
- language, context-free: a language which could be described by a context-free grammar
- language, context-sensitive a language which could be described by a context-sensitive grammar
- logic: a branch of Mathematics studying, describing, modeling, axiomatizing and discussing the concept of truth and consequence
- model: a structure which satisfies a given set of logical formulas; also used in a very ample manner as an example or way of studying or abstracting some concept or question; a concrete realization of some given idea
- normal form: a term or object which cannot be reduced anymore
- normalization, strong (SN): the non existence of infinite derivations; it applies to an object as well as to a set of objects
- normalization, weak (WN): the existence of a derivation leading to a normal form; it applies to an object as well as to a set of objects
- order: a relation satisfying reflexivity, symmetry and transitivity
- order, total: an order which additionally satisfies that given any pair of elements, they are related in one way or the other
- orthogonal: for a TRS, the property of not having critical pairs and being left-linear
- parallelization: a technique to proving confluence consisting in considering a rule which rewrites in a single step all possible redexes which are parallel, i.e. they do not overlap or interfere with each other

- pattern: a term (or syntactical description) which may allow different terms to match; also used as a skeleton or structure description
- pattern matching: a technique which allows to describe a function using patterns; the possibility for a term to be an instance of a pattern, i.e. to be the result of some substitution applied to that pattern
- perpetuality: the possibility of following an infinite reduction
- position: a string (usually consisting of natural numbers) indicating the location (as a branch) of a given sub-term within a given term
- Post canonical system (PCS): an ARS where terms are the strings for a given (finite) alphabet and rules are pairs of strings, possibly using variables which denote strings
- problem, decidable: a problem for which an algorithm exists for solving every instance of it
- problem, undecidable: a problem which is not decidable, i.e. there does not exist any algorithm for solving every instance of it
- program: an implemented algorithm, an algorithm specified using a programming language in such a way that can be executed by a computer
- redex: a term which can be rewritten, or a term with a position indicating the corresponding sub-term which will be rewritten
- reduct: a term which comes from another term which has been rewritten; the result of a reduction applied to a given term
- reduction: in rewriting, the transformation of an object into another by using a rewriting relation
- reflexive relation: a relation which satisfies that every element is related with itself
- relation: a set of pairs of objects from a given set; a way or procedure for classifying pairs of objects
- rewriting: a mathematical theory which studies and describes transformations between syntactical objects over general or specific languages
- rule, parallel: a rule which reduces many redexes in a single step, namely all which can be reduced without overlap or interference
- rule, rewrite: a portion of information on how to rewrite

- semantics: the theory which studies and describes the meaning of the language structures
- semi-Thue rewriting system (STS): an ARS where terms are the strings for a given (finite) alphabet and rules are pairs of strings
- set theory: a mathematical theory which tries to model, axiomatize and prove questions about membership into sets (classes, collections)
- simulation: a derivation which calculates the same term that another one; a property which guarantees that, starting from any term, there exists a derivation with the same final reduct than another, the one which is simulated
- skeleton: a structure denoting a term, perhaps a term with some of its information deleted; a skeleton may indicate which class the term is or which structure it may have, but without specific contents
- soundness: correctness, consistency with a given calculus or theory, preservation of classical reductions
- sort: a type, a family of terms; it is commonly used for an n-sorted algebra, for instance a two-sorted calculus (like $\lambda\sigma$)
- standard: sometimes, a normal form; a strategy which reaches normal forms
- standardization: a technique or procedure to achieve, for any given element, a normal form
- step, rewriting: an ordered pair of elements from a rewriting system where the first one rewrites to the second
- strategy, rewriting: a criterion on how to rewrite every possible term admitting rewrite; a deterministic subsystem of a given ARS (sometimes, any sub-ARS)
- strategy, effective: a strategy which is algorithmic, i.e. there is a procedure which calculates the reduct of any given term admitting reduction
- strategy, perpetual: a strategy that always reduces along an infinite derivation if it exists
- string: a sequence (ordered set) of characters
- string rewriting system: a rewriting system whose objects are strings and its rules are pairs of strings
- subject reduction: the preservation of typing after reduction

- substitution: a function (or procedure) intended to substitute, inside a given term, every occurrence of a given variable by another given term
- substitution, closed/ground: an explicit substitution which do not contain (meta-)variables
- substitution, explicit: a syntactical element (as a part of a given term) denoting a substitution; also, the techniques and study of calculi which allow the use of these elements
- substitution, implicit: a function denoting a substitution to do in a given term
- substitution, open: an explicit substitution which may contain (meta-)variables (which may denote terms or substitutions)
- sub-string: a string which occurs inside another string
- sub-term: a term as part of another term; a term occurring inside another term in a given position
- sub-tree: a tree which occurs inside another tree, for example a branch
- symmetrical relation: a relation which satisfies that if a is related to b then b is related to a
- syntax: the theory which studies and describes the superficial structure of the languages
- term: a syntactical object or construction coming from a signature, to which rewrite rules may be applied
- term, closed/ground: a term which cannot contain (meta-)variables
- term, open: a term which can contain (meta-)variables
- term, semi-open: a term which may contain (meta-)variables of sort term but not of sort substitution
- termination: normalization, the halting of a process or calculation
- term rewriting system (TRS): an ARS where the terms are first order terms of a signature, and rules satisfy some properties: left- hand sides should not be variables, and right-hand side variables should also appear in the corresponding left-hand sides
- theory: a set of mathematical (usually logical) formulas which is closed under inference rules; in λ-calculus a theory usually consists of a set of equalities which are closed under β-equivalence

- transitive relation: a relation which satisfies that if a related to b and b related to c then a is related to c
- translation: a mapping from one universe to another, for example mapping the the syntax of a calculus into another one
- tree: a hierarchical structure of information, an acyclic connected directed graph
- type: in λ-calculi, a way of classifying some terms, usually according to the arguments to which they can be applied and to the return values
- typing: the possibility of assigning a type to a given term
- undecidability: the non-existence of an algorithm to solve a given problem
- unification: the possibility that two terms (or patterns) could become equal by applying to them some substitution
- variable: a term which cannot be decomposed into sub-terms different than itself, and it is not a constant; a syntactical object intended to be replaced by some other possible objects
- variable, bound: a variable inside a term which is related to a variable which is marked, for example, by an abstractor
- variable, free: a variable inside a term which is not bound
- zoom-in: a strategy which looks inside a term for another sub-term which verifies some property


Si hay algo que sé bailar es la lambada
y yo la Jota, aunque de reescritura ... ni Jota
I hate to hang up
Rock ∧ Roll
a sound calculus !
&
classical λ-calculus music
I am semi-open to you
I'll not use substitutions
I'll be λxy.x
1 2 TRS
Yo soy la clave de todo
La música es un lenguaje universal ... libre de ... todo contexto.

Appendix C

Some information about this document

La niñez y el genio tienen en común el mismo órgano motor: la curiosidad. – F. Bacon

Nunca sabes quién tiene razón, pero siempre sabes quién manda. – Anónimo

This thesis collects the first formal incursion by the author in the territories of rewriting theory, namely in explicit substitution. We provide here some minimal data related to this thesis: the information contained herein, accessibility, some software tools employed, possible errata and other issues.

C.1 Home page

The internet web page which accompanies this document is located at the URL http://www.dc.uba.ar/people/materias/reescritura/Ariel/PhDthesis. In this page, additional data, exercises, software, comments, possible errata and addenda eventually appear for reference.

C.2 Software

This PhD thesis was written using among other the following PC software packages:

- the implementation MikTex 2.4 of the LaTeX mathematical text processor
- the WinEdit 5.4 LaTeX editor
- parsing utilities by the author in order to preprocess macros, files and indices.

C.3 Copyright questions

This document, except when explicitly indicated, has been entirely produced and written by the author. The intellectual property of each article corresponds to him (and possibly to coauthors where indicated) and to the University of Buenos Aires, according to current rules and regulations for thesis. Any reproduction (rewriting?) of all or parts of the document is prohibited, unless the author is contacted for authorization. Nevertheless all the information can be used in further work or research, if cited properly.

C.4 The author

Some brief information about the author can be found at the internet URL address http://www.dc.uba.ar/people/materias/reescritura/Ariel (although it might be outdated).

Areas of interest within Computer Science are: logic, rewriting, lambda calculus, compiler design, game theory.

Is it raining outside?

Index

No hay afirmación tan absurda que un filósofo no sea capaz de hacer. – M. T. Cicerón

Para tener verdadera libertad hay que ser esclavo de la filosofía. – Epicuro

El progreso de la ciencia es inversamente proporcional a la cantidad de monografías que se publican. – Murphy

Toda solución genera nuevos problemas. – Murphy

Los buenos escritores tienen estas dos cosas en común: prefieren ser comprendidos a ser admirados, y no escriben para el lector demasiado astuto y demasiado crítico. – F. Nietzsche

Nada escribe aquél cuyos escritos no se leen. – Marcial

I've travelled every country,
I've travelled in my mind
It seems we're on a journey,
A trip through space and time
And somewhere lies the answer
To all the questions why
What really makes the difference
Between all dead and living things,
The will to stay alive.
– ABBA, *Move On*